Knobbe Martens Olson & Bear LLP

Intellectual Property Law

2040 Main Street
Fourteenth Floor
Irvine, CA 92614
Tel 949-760-0404
Fax 949-760-9502
knob.com

ORIGINAL WILL FOLLOW VIA:

- ☐ MAIL
- ☐ INTERNATIONAL AIRMAIL
- ☐ COURIER
- ■ WILL NOT FOLLOW
- ☐ HAND DELIVERY
- ☐ WITH ENCLOSURES
- ☐ WITHOUT ENCLOSURES

Facsimile Transmittal Sheet

Confidentiality Notice:

TO: Examiner Kuen S. Lu, Group Art Unit 2177

FIRM: UNITED STATES PATENT AND TRADEMARK OFFICE

FACSIMILE NO: 703-746-9694.

YOUR REFERENCE: U.S. Patent Application No. 10/050,675 OUR REFERENCE: FNDSTN.013A

FROM: Ted M. Cannon, Reg. No. 55,036

TOTAL NUMBER OF PAGES: **104** (INCLUDING COVER SHEET). **PART 2 OF 2**

OPERATOR: Kristin Eldred DATE: July 1, 2004 TIME:

IF YOU DID NOT RECEIVE ALL OF THE PAGES, PLEASE CALL 949-721-2849 IMMEDIATELY.

MESSAGE: Dear Examiner Lu:

In accordance with our telephone conference. I am attaching copies of the documents you requested. Please call Jerry T. Sewell at 949-721-2849 or Kristin Eldred at 949-721-2846 if you need additional information.

JTS-20289.DOC:ke
20040630

550 West C Street
Suite 1200
San Diego CA 92101
Tel 619-235-8550

201 California Street
Suite 1150
San Francisco CA 94111
Tel 415-954-4114

1901 Avenue of the Stars
Suite 1500
Los Angeles CA 90067
Tel 310-551-3450

3403 Tenth Street
Suite 700
Riverside CA 92501
Tel 909-781-9231

1114 Marsh Street
San Luis Obispo CA 93401
Tel 805-547-5580
Fax 805-547-5590

and fragmentation is not allowed (the DF Bit is set), than the size of the MTU used should be lowered. There is no active measures with Sun Solaris as far as I know.

This is a simple operating system fingerprinting method, which does not require additional or unusual patterns to be set.

The following operating systems where queries and checked for this kind of behavior: Linux Kernel 2.4 test 2,4,5,6; Linux Kernel 2.2.x; FreeBSD 4.0, 3.4; OpenBSD 2.7,2.6; NetBSD 1.4.1,1.4.2; BSDI BSD/OS 4.0,3.1; Solaris 2.6,2.7,2.8; HP-UX 10.20, 11.0x; Compaq Tru64 5.0; Aix 4.1,3.2; Irix 6.5.3, 6.5.8; Ultrix 4.2 – 4.5; OpenVMS v7.1-2; Novel Netware 5.1 SP1, 5.0, 3.12; Microsoft Windows 98/98SE/ME, Microsoft Windows NT WRKS SP6a, Microsoft Windows NT Server SP4, Microsoft Windows 2000 Family.

6.2.1 Avoidance

With Sun Solaris and HPUX operating systems we can use a configuration option in order not to use the DF bit with the ICMP Query replies[31]. With HP-UX 10.30 and 11.x it would not allow the Path MTU Discovery process with ICMP Query replies to be done. This would avoid the fingerprinting method I have introduced, which is based on the fact the DF bit is set on ICMP Query replies from Sun Solaris, and HP-UX 10.30, and 11.x.

With HP-UX 10.30, & 11.0[32], one of the ndd command option is the ip_pmtu_strategy. The variable settings for this option are either 1 or 2. If this bit value is 2, than the Path MTU Discovery Process is used with ICMP Echo Requests. This is the default value. If this bit value equals 1, than the HP-UX machines will not use the ICMP echo-request PMTU discovery strategy, and will not set the DF bit after determining the accurate PMTU.

To turn off ip_path_mtu_discovery on a Sun Solaris machine use the following command as root:

```
# ndd -set  /dev/ip  ip_path_mtu_discovery 0
```

Than when the ICMP Echo Reply is sent (this example) the DF bit is not set:

```
# SING v1.0beta7 initiated on Host_Address at Thu Sep 14 10:01:02 2000
# Command line:
# -> sing -c 1 -L Host_Address
SINGing to Host_Address (IP_Address): 16 data bytes
16 bytes from 10.13.57.20: icmp_seq=0 ttl=254 TOS=0 time=1.578 ms

--- Host_Address sing statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 1.578/1.578/1.578 ms
# SING finished at Thu Sep 14 10:01:02 2000
```

This was tested against Solaris 2.5.1, Solaris 2.6 and Solaris 2.7, all SPARC boxes.

Beware - With Sun Solaris turning this option off, will turn off the PMTU discovery process with TCP as well. This is not recommended because of performance issues.

---

[31] I do not have any information regarding AIX.

[32] Building a Bastion Host Using HP UX 11, Kevin Stevens, http://people.hp.se/stevesk/bastion11.html.

6.3 The IP Time-to-Live Field Value with ICMP

The sender sets the time to live field to a value that represents the maximum time the datagram is allowed to travel on the Internet.

The field value is decreased at each point that the Internet header is being processed. RFC 791 states that this field decreasement reflects the time spent processing the datagram. The field value is measured in units of seconds. The RFC also states that the maximum time to live value can be set to 255 seconds, which equals 4.25 minutes. The datagram must be discarded if this field value equals zero - before reaching its destination.

Relating to this field as a measure to assess time is a bit misleading. Some routers may process the datagram faster than a second, and some may process the datagram longer than a second.

The real intention is to have an upper bound to the datagrams lifetime, so infinite loops of undelivered datagrams will not jam the Internet.

Having a bound to the datagram's lifetime help us to prevent old duplicates to arrive after a certain time elapsed. So when we retransmit a piece of information which was not previously delivered we can be assured that the older duplicate is already discarded and will not interfere with the process.

The IP TTL field value with ICMP has two separate values, one for ICMP query messages and one for ICMP query replies.

The TTL field value helps us identify certain operating systems and groups of operating systems. It also provides us with the simplest means to add another check criteria when we are querying other host(s) or listening to traffic (sniffing).

6.3.1 IP TTL Field Value with ICMP Query Replies

We can use the IP TTL field value with the ICMP Query Reply datagrams to identify certain groups of operating systems. The method discussed in this section is a very simple one. We send an ICMP Query request message to a host. If we receive a reply, we would be looking at the IP TTL field value in the ICMP query reply. The next table describes the IP TTL field values with ICMP Echo replies for various operating systems. According to the table we can distinguish between certain operating systems:

Operating System	IP TTL on ICMP datagrams - In Reply -
LINUX Kernel 2.4	255
Kernel 2.2.14	255
Kernel 2.0.x[33]	64
FreeBSD 4.0	255
FreeBSD 3.4	255
OpenBSD 2.7	255
OpenBSD 2.6	255
NetBSD	255
BSDI BSD/OS 4.0	255
BSDI BSD/OS 3.1	255

[33] Stephane Omnes provided information about LINUX Kernel 2.0.x.

Operating System	IP TTL on ICMP datagrams - In Reply -
Solaris 2.5.1	255
Solaris 2.6	255
Solaris 2.7	255
Solaris 2.8	255
HP-UX v10.20	255
HP-UX v11.0	255
Compaq Tru64 v5.0	64
Irix 6.5.3	255
Irix 6.5.8	255
AIX 4.1	255
AIX 3.2	255
ULTRIX 4.2 – 4.5	255
OpenVMS v7.1-2	255
Windows 95	32
Windows 98	128
Windows 98 SE	128
Windows ME	128
Windows NT 4 WRKS SP 3	128
Windows NT 4 WRKS SP 6a	128
Windows NT 4 Server SP4	128
Windows 2000 Family	128

Table 5: IP TTL Field Values in replies from Various Operating Systems

If we would look at the ICMP Echo replies IP TTL field values than we can identify a few patterns:

- UNIX and UNIX-like operating systems use 255 as their IP TTL field value with ICMP query replies.
- Compaq Tru64 5.0 and LINUX 2.0.x are the exception, using 64 as its IP TTL field value with ICMP query replies.
- Microsoft Windows operating system based machines are using the value of 128.
- Microsoft Windows 95 is the only Microsoft operating system to use 32 as its IP TTL field value with ICMP query messages, making it unique among all other operating systems as well.

With the ICMP query replies we have an operating system that is clearly distinguished from the other - Windows 95. Other operating systems are grouped into the " 64 group" (LINUX based Kernel 2.0.x machines & Compaq Tru64 5.0), the "255 group" (UNIX and UNIX-like), and into the "128 group" (Microsoft operating systems).

We are not limited to ICMP ECHO replies only. We can use the other ICMP Query message types as well, and the results should be the same. In the next example an ICMP Timestamp request is sent to a Redhat 6.1 LINUX, Kernel 2.2.12 machine:

```
[root@stan /root]# icmpush -tstamp 192.168.5.5
kenny.sys-security.com -> 13:48:07
```

The Snort Trace:

```
01/26-13:51:29.342647 192.168.5.1 -> 192.168.5.5
ICMP TTL:254 TOS:0x0 ID:13170
TIMESTAMP REQUEST
88 16 D8 D9 02 8B 63 3D 00 00 00 00 00 00 00 00  ......c=........

01/26-13:51:29.342885 192.168.5.5 -> 192.168.5.1
ICMP TTL:255 TOS:0x0 ID:6096
TIMESTAMP REPLY
88 16 D8 D9 02 8B 63 3D 02 88 50 18 02 88 50 18  ......c=..P...P.
2A DE 1C 00 A0 F9                                *.....
```

IP TTL field value is 255 (the machine is on the same LAN).

We can use this information with other tests as, to provide us extra criteria with zero effort.

6.3.2 IP TTL Field Value with ICMP ECHO Requests

The examination of the IP TTL field value is not limited to ICMP Query replies only. We can learn a lot from the ICMP requests as well. The following is a Table summarizing various operating system default values for the IP TTL field embedded inside an ICMP Query request:

Operating System	IP TTL on ICMP datagrams - In Reply -	IP TTL on ICMP datagrams - In Req. -
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	255	64
Redhat LINUX 6.2 Kernel 2.2.14	255	64
LINUX Kernel 2.0.x	64	64
FreeBSD 4.0	255	255
FreeBSD 3.4	255	255
OpenBSD 2.7	255	255
OpenBSD 2.6	255	255
NetBSD	255	
BSDI BSD/OS 4.0	255	
BSDI BSD/OS 3.1	255	
Solaris 2.5.1	255	255
Solaris 2.6	255	255
Solaris 2.7	255	255
Solaris 2.8	255	255
HP-UX v10.20	255	255
HP-UX v11.0	255	
Compaq Tru64 v5.0	64	
Irix 6.5.3	255	
Irix 6.5.8	255	
AIX 4.1	255	
AIX 3.2	255	
ULTRIX 4.2 – 4.5	255	

Operating System	IP TTL on ICMP datagrams - In Reply -	IP TTL on ICMP datagrams - In Req. -
OpenVMS v7.1-2	255	
Windows 95	32	32
Windows 98	128	32
Windows 98 SE	128	32
Windows ME	128	32
Windows NT 4 WRKS SP 3	128	32
Windows NT 4 WRKS SP 6a	128	32
Windows NT 4 Server SP4	128	32
Windows 2000 Professional	128	128
Windows 2000 Server	128	128

Table 6: IP TTL Field Values in requests from Various Operating Systems

The ICMP Query message type used was ICMP Echo request, which is common on all operating systems tested using the ping utility.

- LINUX Kernel 2.0.x, 2.2.x & 2.4.x use 64 as their IP TTL Field Value with ICMP Echo Requests.
- FreeBSD 4.1, 4.0, 3.4; Sun Solaris 2.5.1, 2.6, 2.7, 2.8; OpenBSD 2.6, 2.7, NetBSD and HP UX 10.20 use 255 as their IP TTL field value with ICMP Echo requests. With the OSs listed above the same IP TTL Field value with any ICMP message is given.
- Windows 95/98/98SE/ME/NT4 WRKS SP3,SP4,SP6a/NT4 Server SP4 - all using 32 as their IP TTL field value with ICMP Echo requests.
- A Microsoft window 2000 is using 128 as its IP TTL Field Value with ICMP Echo requests.

We can distinguish between LINUX, Microsoft Windows 2000, the other Microsoft operating systems group, and the "255 group" using this method.

6.3.3 Correlating the Information

Using the IP TTL field value with ICMP messages we can distinguish between Microsoft Windows 2000, certain Microsoft Windows Operating systems, LINUX Kernel 2.2.x & 2.4.x, LINUX Kernel 2.0.x, and the *BSD and Solaris group.

Operating System	IP TTL value in the ECHO Requests	IP TTL value in the ECHO Replies
Microsoft Windows Family	32	128
*BSD and Solaris	255	255
LINUX Kernel 2.2.x & 2.4.x	64	255
LINUX Kernel 2.0.x	64	64
Microsoft Windows 2000	128	128
Microsoft Windows 95	33	32

Table 7: Further dividing the groups of operating systems according to IP TTL field value in the ICMP ECHO Requests and in the ICMP ECHO Replies

One would expect that the IP TTL field value would be the same with both ICMP Query requests and ICMP Query replies. Apparently this is not true and provide us with valuable information about the operating system querying / being queried.

6.4 Using Fragmented ICMP Address Mask Requests (Identifying Sun Solaris & HP-UX 11.0x machines)[34].

It appears that only some of the operating systems would answer an ICMP Address Mask Request as it is outlined in Table 2 in section 2.5. Those operating systems include - ULTRIX OpenVMS, Windows 95/98/98 SE/ME, NT below SP 4, HP-UX 11.0x and SUN Solaris. How can we distinguish between those who answer the request?

This is a regular ICMP Address Mask Request sent by SING to a SUN Solaris 2.7 machine:

```
[root@aik icmp]# ./sing -mask IP_Address
SINGing to IP_Address (IP_Address): 12 data bytes
12 bytes from IP_Address: icmp_seq=0 ttl=236 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=1 ttl=236 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=2 ttl=236 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=3 ttl=236 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=4 ttl=236 mask=255.255.255.0

--- IP_Address sing statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
```

All operating systems that would answer with ICMP Address Mask Reply would reply with the Address Mask of the network they reside on.

What would happen if we would introduce a little twist? Lets say we would send those queries fragmented?

In the next example, I have sent ICMP Address Mask Request to the same SUN Solaris 2.7 box, this time fragmented to pieces of 8 bytes of IP data. As we can see the answer I got was unusual:

```
[root@aik icmp]# ./sing -mask -c 2 -F 8 IP_Address
SINGing to IP_Address (IP_Address): 12 data bytes
12 bytes from IP_Address: icmp_seq=0 ttl=241 mask=0.0.0.0
12 bytes from IP_Address: icmp_seq=1 ttl=241 mask=0.0.0.0

--- IP_Address sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
[root@aik icmp]#
```

The tcpdump trace:

```
20:02:48.441174 ppp0 > y.y.y.y > Host_Address: icmp: address mask
request (frag 13170:8@0+)
                        4500 001c 3372 2000 ff01 50ab yyyy yyyy
                        xxxx xxxx 1100 aee3 401c 0000
20:02:48.442858 ppp0 > y.y.y.y > Host_Address: (frag 13170:4@8)
                        4500 0018 3372 0001 ff01 70ae yyyy yyyy
                        xxxx xxxx 0000 0000
```

[34] The Solaris portion was also discovered by Alfredo Andres Omella.

```
20:02:49.111427 ppp0 < Host_Address > y.y.y.y: icmp: address mask is
0x00000000 (DF)
                        4500 0020 3618 4000 f101 3c01 xxxx xxxx
                        yyyy yyyy 1200 ade3 401c 0000 0000 0000

20:02:49.441492 ppp0 > y.y.y.y > Host_Address: icmp: address mask
request (frag 13170:8@0+)
                        4500 001c 3372 2000 ff01 50ab yyyy yyyy
                        xxxx xxxx 1100 ade3 401c 0100
20:02:49.442951 ppp0 > y.y.y.y > Host_Address: (frag 13170:4@8)
                        4500 0018 3372 0001 ff01 70ae yyyy yyyy
                        xxxx xxxx 0000 0000
20:02:50.011433 ppp0 < Host_Address > y.y.y.y: icmp: address mask is
0x00000000 (DF)
                        4500 0020 3619 4000 f101 3c00 xxxx xxxx
                        yyyy yyyy 1200 ace3 401c 0100 0000 0000
```

The same SUN Solaris box now replies with a 0.0.0.0 as the Address Mask for the Network it resides on. The same behavioral patterns were produced against an HP-UX 11.0x operating system based machine[35].

What would happen with the other operating systems?

They all would respond with the real Address Mask in their replies.

Here we got a distinction between SUN Solaris & HP-UX 11.0x based machines to the other operating systems that would answer those queries.

In Section 6.2 we were discussing the various issues regarding the DF Bit usage within the ICMP Query replies. Both SUN Solaris and HP-UX 11.0x set the DF Bit by default in their ICMP Query replies. HP-UX 11.0x based machines starts setting the DF Bit after they finishes the ICMP based PMTU Discovery process (Querying the Host that has queried them with ICMP Echo request) – if it is enabled by default. This gives us another means to distinguish between those two operating systems on top of another method.

We can further try to distinguish between the remaining operating systems. This, if we would use the I=0 code method I am going to introduced in section 6.8.

Important notice: When I have tested this method I have encountered some problems replicating the results with different ISPs. As it seems from analyzing the information I got, certain ISPs would block fragmented ICMP datagrams. This behavior would not enable this method to succeed. One way of testing this is to send a regular ICMP Echo request. We should watch for a response from the probed machine. If received, than we should send ICMP Echo request, this time fragmented. If no reply is received than your ISP is blocking ICMP fragments probably.

[35] When I have published this information in Bugtraq (August 5, 2000) Peter J. Holzer notified me that HP-UX 11.00 produce the same behavior as the SUN Solaris boxes. Darren Reed also noted that because SUN Solaris and HP-UX 11.0 share the same third party (Mentat) implementation for some of their TCP/IP stacks this behavior is produced by both.

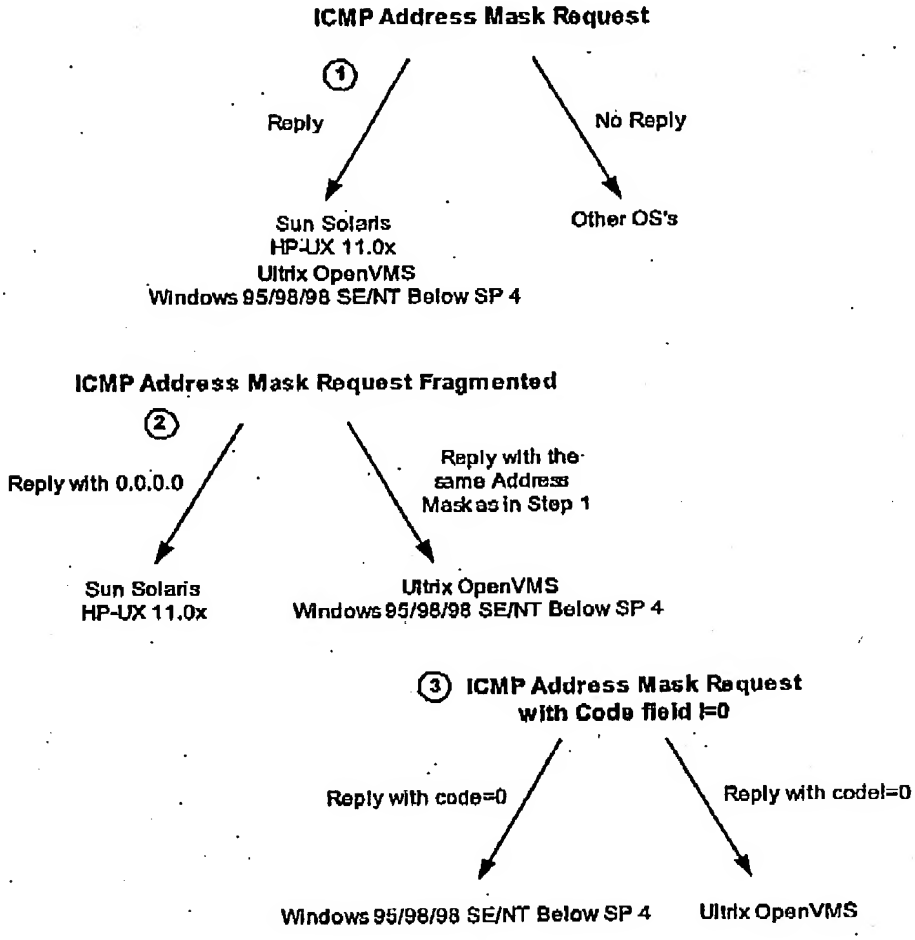

Diagram 5: Finger Printing Using ICMP Address Mask Requests

Using Crafted ICMP Query Messages

Playing with the TOS Field

Each IP Datagram has an 8-bit field called the "TOS Byte", which represents the IP support for prioritization and Type-of-Service handling.

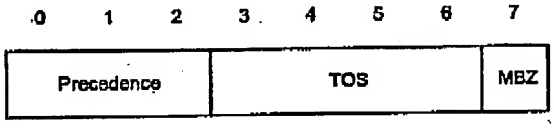

Figure 11: The Type of Service Byte

The "TOS Byte" consists of three fields.

The "Precedence field", which is 3-bit long, is intended to prioritize the IP Datagram. It has eight levels of prioritization[36]:

Precedence	Definition
0	Routine (Normal)
1	Priority
2	Immediate
3	Flash
4	Flash Override
5	Critical
6	Internetwork Control
7	Network control

Table 8: Precedence Field Values

Higher priority traffic should be sent before lower priority traffic.

The second field, 4 bits long, is the "Type-of-Service" field. It is intended to describe how the network should make tradeoffs between throughput, delay, reliability, and cost in routing an IP Datagram.

RFC 1349[37] has defined the "Type-of-Service" field as a single enumerated value, thus interpreted as a numeric value rather than independent flags (with RFC 791 the 4 bits were distinct options, allowing combinations as well). The 4 bits represents a maximum of 16 possible values.

Value (Hex)	Value (Dec)	Value (Binary)	Service
0	0	0000	Normal
1	1	1000	Minimize Delay
2	2	0100	Maximize Throughput
4	4	0010	Maximize Reliability
8	8	0001	Minimize Cost
F	15	1111	Maximize Security[38]

Table 9: Type-of-Service Field Values

What about the other 10 value possibilities?

RFC 1349 refer to this issue and states that "although the semantics of values other than the five listed above are not defined by this memo, they are perfectly legal TOS values, and hosts and routers must not preclude their use in any way"..."A host or a router need not make any distinction between TOS values who's semantics are defined by this memo and those that are not".

[36] RFC 791 – Internet Protocol, http://www.ietf.org/rfc/rfc791.txt.
[37] RFC 1349 - Type of Service in the Internet Protocol Suite, http://www.ietf.org/rfc/rfc1349.txt.
[38] RFC 1455 - Physical Link Security Type of Service, http://www.ietf.org/rfc/rfc1455.txt.

The last field, the "MBZ" (must be zero), is unused and must be zero. Routers and hosts ignore this last field. This field is 1 bit long.

Combining Type-of-Service flags with the different prioritization values, dictates very explicit types of behavior with certain types of data.

Please note the not all TCP/IP implementations would use this values (nor offer a mechanism for setting those values) and some will not handle datagrams which have Type-of-Service and/or Precedence values other than the defaults, differently.

6.5 Precedence Bits Echoing (Fingerprinting Microsoft Windows 2000, ULTRIX, HPUX 11.0&10.30, OpenVMS and more)

The precedence bits behavior is a problem. RFC 1122, which defines the requirements for Internet Hosts, does not outline the way to handle the Precedence Bits with ICMP. The RFC only statement about the Precedence Bits is:

> "The Precedence field is intended for Department of Defense applications of the Internet protocols. The use of non-zero values in this field is outside the scope of this document and the IP standard specification. Vendors should consult the Defense Communication Agency (DCA) for guidance on the IP Precedence field and its implications for other protocol layers. However, vendors should note that the use of precedence will most likely require that its value be passed between protocol layers in just the same way as the TOS field is passed".

This does not give us something to work with.

RFC 1812, Requirements for IP version 4 routers state that:

> "An ICMP reply message MUST have its IP Precedence field set to the value as the IP Precedence field in the ICMP request that provoked the reply".

Echoing back the Precedence field value has its logic, because the TOS field should be echoed back with an ICMP Query replies, and both the Precedence field and the TOS field were to dictate very explicit types of behavior with certain types of data.

As you can see we do not have a clear ruling about this issue. I was thinking it might be a ground for an operating system fingerprinting method.

Most operating systems I have checked will behave as the next behavioral example with AIX 4.3. With this example an ICMP Echo request is sent which carries a value for the TOS field.

```
[root@godfather precedence_echo]# /usr/local/bin/sing -c 5 -TOS 128
y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=239 TOS=128 time=5896.472 ms
16 bytes from y.y.y.y: seq=1 ttl=239 TOS=128 time=5952.071 ms
16 bytes from y.y.y.y: seq=2 ttl=239 TOS=128 time=6102.020 ms
16 bytes from y.y.y.y: seq=3 ttl=239 TOS=128 time=6261.997 ms
16 bytes from y.y.y.y: seq=4 ttl=239 TOS=128 time=5842.726 ms

--- y.y.y.y sing statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
```

```
round-trip min/avg/max = 5842.726/6011.057/6261.997 ms
[root@godfather precedence_echo]#
```

The tcpdump trace:

```
21:02:53.241666 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x80]
(ttl 255, id 13170)
                  4580 0024 3372 0000 ff01 619c xxxx xxxx
                  yyyy yyyy 0800 c278 6f05 0000 dd97 0d3a
                  d8af 0300

21:02:59.134297 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x80]
(ttl 239, id 40656)
                  4580 0024 9ed0 0000 ef01 063e yyyy yyyy
                  xxxx xxxx 0000 ca78 6f05 0000 dd97 0d3a
                  d8af 0300
```

The Host queried is using the value used for the ICMP Echo Request with its ICMP Echo Reply.

Some operating systems are the exception.

The next example is with Microsoft Windows 2000. The same ICMP Echo Request was sent.

```
[root@godfather precedence_echo]# /usr/local/bin/sing -c 5 -TOS 128
y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=111 TOS=0 time=6261.043 ms
16 bytes from y.y.y.y: seq=1 ttl=111 TOS=0 time=6422.019 ms
16 bytes from y.y.y.y: seq=2 ttl=111 TOS=0 time=6572.675 ms
16 bytes from y.y.y.y: seq=4 ttl=111 TOS=0 time=6282.022 ms

--- y.y.y.y sing statistics ---
5 packets transmitted, 4 packets received, 20% packet loss
round-trip min/avg/max = 6261.043/6384.440/6572.675 ms
[root@godfather precedence_echo]#
```

The tcpdump trace:

```
20:13:36.717070 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x80]
(ttl 255, id 13170)
                  4580 0024 3372 0000 ff01 d95d xxxx xxxx
                  yyyy yyyy 0800 df43 c304 0000 508c 0d3a
                  edf0 0a00

20:13:42.974295 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply (ttl 111, id
26133)
                  4500 0024 6615 0000 6f01 373b yyyy yyyy
                  xxxx xxxx 0000 e743 c304 0000 508c 0d3a
                  edf0 0a00
```

The ICMP Echo Reply will not use the value assigned to the Precedence Bits with the ICMP Echo Request.

Which operating systems share this behavioral pattern? Microsoft Windows 2000 Family, and ULTRIX.

Differentiating between Microsoft Windows 2000 and Ultrix is easily achieved if we examine the IP TTL field value. With ULTRIX the value assigned to the ICMP Echo reply will be 255, with Microsoft Windows 2000 it will be 128.

Another interesting case is with HPUX 11.0. Lets examine the trace and logs:

```
[root@godfather precedence_echo]# /usr/local/bin/sing -c 2 -TOS 128
y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=242 TOS=128 time=639.274 ms
16 bytes from y.y.y.y: seq=1 DF! ttl=242 TOS=0 time=310.427 ms

--- y.y.y.y sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 310.427/474.850/639.274 ms
```

The first reply from the HPUX machine echoed back the TOS field value we were using with the ICMP Echo Request. But what have happened between the first and the second reply?

```
00:35:09.315260 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x80]
(ttl 255, id 13170)
                        4580 0024 3372 0000 ff01 4bd1 xxxx xxxx
                        yyyy yyyy 0800 16f0 db3c 0000 9dc9 0d3a
                        56cf 0400

00:35:09.944274 ppp0 < y.y.y.y > x.x.x.x: icmp: echo request (DF) (ttl
242, id 22417)
                        4500 05dc 5791 4000 f201 ef79 yyyy yyyy
                        xxxx xxxx 0800 7e52 9abc def0 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
                        0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
```

The first request was sent, as an instant reply the HPUX 11.0 machine started its PMTU discovery process with ICMP Echo Requests and sent an ICMP Echo Request 1500 bytes long[39].

```
00:35:09.944355 ppp0 > x.x.x.x > y.y.y.y: icmp: echo reply (ttl 255, id
14194)
                    4500 05dc 3772 0000 ff01 4299 xxxx xxxx
                    yyyy yyyy 0000 8652 9abc def0 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000 0000 0000
```

[39] For an explanation about the HPUX 11.0 PMTU process using ICMP Echo Requests please see section 6.2.

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
                    0000 0000 0000 0000 0000 0000

00:35:09.954282 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x80]
(ttl 242, id 22418)
                    4580 0024 5792 0000 f201 34b1 yyyy yyyy
                    xxxx xxxx 0000 1ef0 db3c 0000 9dc9 0d3a
                    56cf 0400
```

The ICMP Echo Reply received from the HPUX 11.0 machine for the ICMP Echo Request echoed back the TOS field value.

Another ICMP Echo Request was sent with TOS field value of 0x80 hex:

```
00:35:10.314321 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x80]
(ttl 255, id 13170)
                    4580 0024 3372 0000 ff01 4bd1 xxxx xxxx
                    yyyy yyyy 0800 b7f3 db3c 0100 9ec9 0d3a
                    b3cb 0400

00:35:10.624275 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply (DF) (ttl
242, id 22419)
                    4500 0024 5793 4000 f201 f52f yyyy yyyy
                    xxxx xxxx 0000 bff3 db3c 0100 9ec9 0d3a
                    b3cb 0400
```

The ICMP Echo Reply received did not echo back the TOS field value, and set the DF bit. The PMTU discovery process finished its initial stages and went to regular operation. From now on the ICMP Echo Replies did not echo the TOS field value.

This gives us the ability to track down HPUX 11.0 (and 10.30) machines when they are using the PMTU Discovery process.

6.5.1 Changed Pattern with other ICMP Query Message Types

We can identify change of pattern with OpenVMS, Windows 98, 98SE, and ME. With ICMP Echo replies they all would echo back the TOS field value, but with ICMP Timestamp replies they will change the behavior and send back 0x000. Since OpenVMS use 255 as its IP TTL field value, and the Microsoft Windows based machines use 128, we can differentiate between them and isolate OpenVMS, and the Microsoft based OSs.

Further distinction between the Microsoft operating systems can be achieved if we will query them with ICMP Address Mask request, which only Microsoft Windows 98/98SE will answer for. The Microsoft Windows ME will not reply, enabling us to identify it.

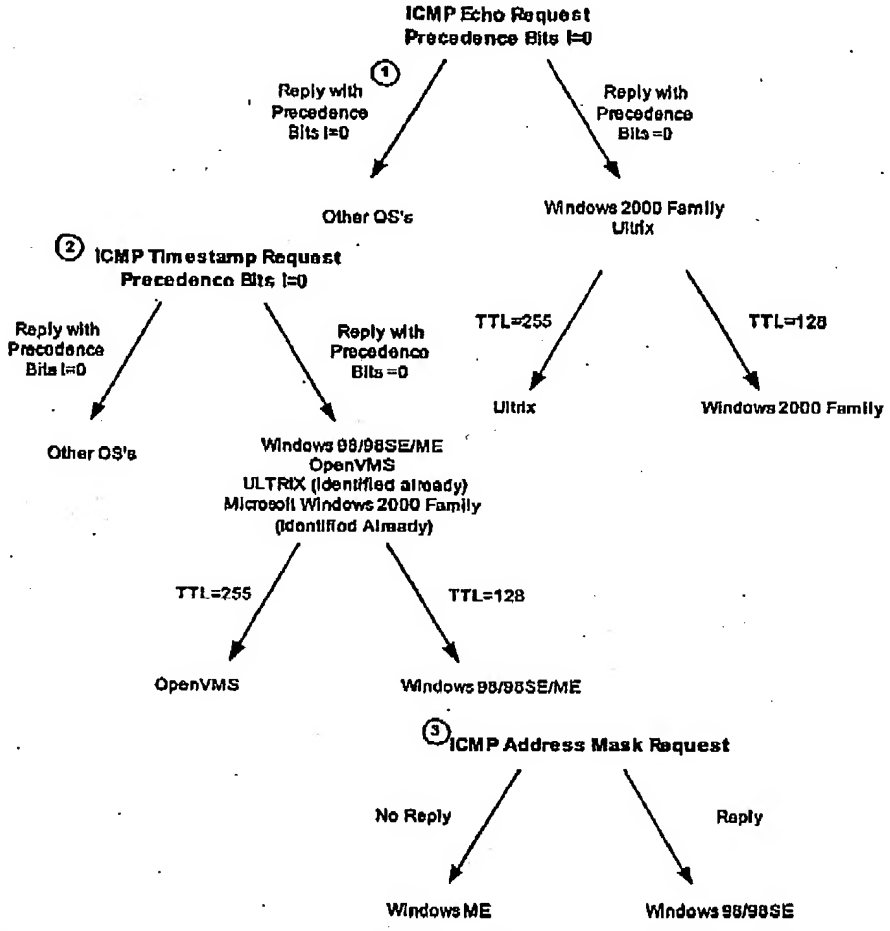

Diagram 6: An example for a way to fingerprint Microsoft Windows 2000, Ultrix, HPUX 11.0 & 10.30, OpenVMS, Microsoft Windows ME, and Microsoft Windows 98/98SE based machines with ICMP Query messages with the Precedence Bits field !=0

Operating System	Information Request With Precedence!=0	Time Stamp Request With Precedence!=0	Address Mask Request With Precedence!=0	Echo Request With Precedence!=0
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	Not Answering	!=0x00	Not Answering	!=0x00
Redhat LINUX 6.2 Kernel 2.2.14	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.0	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.1.1	Not Answering	!=0x00	Not Answering	
OpenBSD 2.7	Not Answering		Not Answering	!=0x00
OpenBSD 2.6	Not Answering		Not Answering	!=0x00
NetBSD	Not Answering		Not Answering	!=0x00
BSDI BSD/OS 4.0	Not Answering		Not Answering	!=0x00

Operating System	Information Request With Precedence!=0	Time Stamp Request With Precedence!=0	Address Mask Request With Precedence!=0	Echo Request With Precedence!=0
BSDI BSD/OS 3.1	Not Answering		Not Answering	!=0x00
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.7	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.8	Not Implemented	!=0x00	!=0x00	!=0x00
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	!=0x00 -> 0x00	!=0x00 -> 0x00
Compaq Tru64 v5.0	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.3	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.2.1	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.1	!=0x00	!=0x00	Not Answering	!=0x00
AIX 3.2	!=0x00	!=0x00	Not Answering	!=0x00
ULTRIX 4.2 – 4.5	0x00	0x00	0x00	0x00
OpenVMS v7.1-2	0x00	0x00	0x00	!=0x00
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x00	0x00	!=0x00
Windows 98 SE	Not Answering	0x00	0x00	!=0x00
Windows ME	Not Answering	0x00	Not Answering	!=0x00
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		!=0x00
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	!=0x00
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	!=0x00
Windows 2000 Professional	Not Answering	0x00	Not Answering	0x00
Windows 2000 Server	Not Answering	0x00	Not Answering	0x00

Table 10: ICMP Query Message Types with Precedence Bits != 0

6.6 TOSing OSs out of the Window / "TOS Echoing" (Fingerprinting Microsoft Windows 2000)

6.6.1 The use of the Type-of-Service field with the Internet Control Message Protocol

RFC 1349 also define the usage of the Type-of-Service field with the ICMP messages. It distinguishes between ICMP error messages (Destination Unreachable, Source Quench, Redirect, Time Exceeded, and Parameter Problem), ICMP query messages (Echo, Router Solicitation, Timestamp, Information request, Address Mask request) and ICMP reply messages (Echo reply, Router Advertisement, Timestamp reply, Information reply, Address Mask reply).

Simple rules are defined:

- An ICMP error message is always sent with the default TOS (0x00)
- An ICMP request message may be sent with any value in the TOS field. "A mechanism to allow the user to specify the TOS value to be used would be a useful feature in many applications that generate ICMP request messages"[40].

[40] RFC 1349 - Type of Service in the Internet Protocol Suite, http://www.ietf.org/rfc/rfc1349.txt.

The RFC further specify that although ICMP request messages are normally sent with the default TOS, there are sometimes good reasons why they would be sent with some other TOS value.

- An ICMP reply message is sent with the same value in the TOS field as was used in the corresponding ICMP request message.

Using this logic I have decided to check if certain operating systems react correctly to an ICMP Query messages with a Type-of-Service field value, which is different than the default (0x00).

The check out was produced with all ICMP query message types sent with a Type-of-Service field set to a known value, than set to an unknown value (the term known and unknown are used here because I was not experimenting with non-legit values, and since any value may be sent inside this field).

The following example is an ICMP Echo request sent to my FreeBSD 4.0 machine with the TOS field equals an 8 hex value, which is a legit TOS value. The tool used here is SING[41]:

```
[root@godfather bin]# ./sing -echo -TOS 8 IP_Address
SINGing to IP_Address (IP_Address): 16 data bytes
16 bytes from IP_Address: icmp_seq=2 ttl=243 TOS=8 time=260.043 ms
16 bytes from IP_Address: icmp_seq=3 ttl=243 TOS=8 time=180.011 ms
16 bytes from IP_Address: icmp_seq=4 ttl=243 TOS=8 time=240.240 ms
16 bytes from IP_Address: icmp_seq=5 ttl=243 TOS=8 time=260.037 ms
16 bytes from IP_Address: icmp_seq=6 ttl=243 TOS=8 time=290.033 ms

--- IP_Address sing statistics ---
7 packets transmitted, 5 packets received, 28% packet loss
round-trip min/avg/max = 180.011/246.073/290.033 ms
[root@godfather bin]#
```

The tcpdump trace:

```
17:23:46.605297 if 4  > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x8]
(ttl 255, id 13170)
                  4508 0024 3372 0000 ff01 60e4 xxxx xxxx
                  yyyy yyyy 0800 0e9a d604 0600 f2ea bc39
                  553c 0900
17:23:46.895255 if 4  < y.y.y.y > x.x.x.x: icmp: echo reply [tos 0x8]
(ttl 243, id 58832)
                  4508 0024 e5d0 0000 f301 ba85 yyyy yyyy
                  xxxx xxxx 0000 169a d604 0600 f2ea bc39
                  553c 0900
```

This is the second test I have produced, sending ICMP Echo request with the Type-of-Service field set to a 10 Hex value, a value that is not a known Type-of-Service value:

[41] SING has the ability to monitor for any replies and than print the received TOS value. I find this option very useful, and thank the author for embedding this function, as I requested.

```
[root@godfather bin]# ./sing -echo -TOS 10 IP_Address
SINGing to IP_Address (IP_Address): 16 data bytes
16 bytes from IP_Address: icmp_seq=0 ttl=243 TOS=10 time=197.933 ms
16 bytes from IP_Address: icmp_seq=1 ttl=243 TOS=10 time=340.048 ms
16 bytes from IP_Address: icmp_seq=2 ttl=243 TOS=10 time=250.025 ms
16 bytes from IP_Address: icmp_seq=3 ttl=243 TOS=10 time=230.019 ms
16 bytes from IP_Address: icmp_seq=4 ttl=243 TOS=10 time=270.017 ms
16 bytes from IP_Address: icmp_seq=5 ttl=243 TOS=10 time=270.017 ms
16 bytes from IP_Address: icmp_seq=6 ttl=243 TOS=10 time=260.021 ms

--- IP_Address sing statistics ---
7 packets transmitted, 7 packets received, 0% packet loss
round-trip min/avg/max = 197.933/259.726/340.048 ms
```

The tcpdump trace:

```
17:24:36.155298 if 4  > y.y.y.y > x.x.x.x: icmp: echo request [tos
0xa,ECT]  (ttl 255, id 13170)
                  450a 0024 3372 0000 ff01 60e2 yyyy yyyy
                  xxxx xxxx 0800 af77 d904 0600 24eb bc39
                  865e 0200
17:24:36.415254 if 4  < x.x.x.x > y.y.y.y: icmp: echo reply [tos
0xa,ECT]  (ttl 243, id 65031)
                  450a 0024 fe07 0000 f301 a24c xxxx xxxx
                  yyyy yyyy 0000 b777 d904 0600 24eb bc39
                  865e 0200
```

As it can be seen from the tcpdump trace, the ICMP Echo reply messages have maintained the Type-of-Service value as was used in the corresponding ICMP request message.

FreeBSD 4.0 does not respond to ICMP Information request, or to ICMP Address Mask requests. I had to verify with ICMP Timestamp requests with the same Type-of-Service values as with the previous ICMP Echo requests that this behavior is produced with ICMP Timestamp request and replies as well.

Again the tool I have used is SING:

```
[root@godfather bin]# ./sing -tstamp -TOS 8 IP_Address
SINGing to IP_Address (IP_Address): 20 data bytes
20 bytes from IP_Address: icmp_seq=0 ttl=243 TOS=8 diff=6832668
20 bytes from IP_Address: icmp_seq=1 ttl=243 TOS=8 diff=6832403
20 bytes from IP_Address: icmp_seq=2 ttl=243 TOS=8 diff=6832633
20 bytes from IP_Address: icmp_seq=3 ttl=243 TOS=8 diff=6832605
20 bytes from IP_Address: icmp_seq=4 ttl=243 TOS=8 diff=6832431

--- IP_Address sing statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
[root@godfather bin]#
```

The tcpdump trace:

```
17:26:00.455295 if 4  > x.x.x.x > y.y.y.y: icmp: time stamp request
[tos 0x8]  (ttl 255, id 13170)
                  4508 0028 3372 0000 ff01 60e0 xxxx xxxx
```

```
                    YYYY YYYY 0d00 345b dd04 0400 0318 da87
                    0000 0000 0000 0000
17:26:00.755254 if 4  < y.y.y.y > x.x.x.x: icmp: time stamp reply [tos
0x8]  (ttl 243, id 5867)
                    4508 0028 16eb 0000 f301 8967 YYYY YYYY
                    xxxx xxxx 0e00 f4ec dd04 0400 0318 da87
                    0380 1bb7 0380 1bb7
```

The second test with TOS field value set to 10 Hex value:

```
[root@godfather bin]# ./sing -tstamp -TOS 10 IP_Address
SINGing to IP_Address (IP_Address): 20 data bytes
20 bytes from IP_Address: icmp_seq=0 ttl=243 TOS=10 diff=6766872
20 bytes from IP_Address: icmp_seq=1 ttl=243 TOS=10 diff=6767059
20 bytes from IP_Address: icmp_seq=2 ttl=243 TOS=10 diff=6767059
20 bytes from IP_Address: icmp_seq=3 ttl=243 TOS=10 diff=6767063
20 bytes from IP_Address: icmp_seq=4 ttl=243 TOS=10 diff=6766892
20 bytes from IP_Address: icmp_seq=5 ttl=243 TOS=10 diff=6766887
20 bytes from IP_Address: icmp_seq=6 ttl=243 TOS=10 diff=6766873
20 bytes from IP_Address: icmp_seq=7 ttl=243 TOS=10 diff=6767057
^C

--- 194.47.250.37 sing statistics ---
9 packets transmitted, 9 packets received, 0% packet loss
[root@godfather bin]#
```

The tcpdump trace:

```
17:25:42.548597 if 4  > x.x.x.x > y.y.y.y: icmp: time stamp request
[tos 0xa,ECT]  (ttl 255, id 13170)
                    450a 0028 3372 0000 ff01 60de xxxx xxxx
                    YYYY YYYY 0d00 7f4e dc04 0000 0318 9494
                    0000 0000 0000 0000
17:25:42.795254 if 4  < y.y.y.y > x.x.x.x: icmp: time stamp reply [tos
0xa,ECT]  (ttl 243, id 3519)
                    450a 0028 0dbf 0000 f301 9291 YYYY YYYY
                    xxxx xxxx 0e00 cbf6 dc04 0000 0318 9494
                    037f d5ac 037f d5ac
```

The same behavior was produced. The ICMP Timestamp replies were sent with the TOS field value equals the TOS filed value of the ICMP Timestamp requests.

Ok. I was curious again. I imagined that the Microsoft Windows implementation of the things might be a little different.

When I was examining ICMP Echo requests I noticed something is wrong with Microsoft:

```
[root@godfather bin]# ./sing -echo -TOS 8 Host_Address
SINGing to Host_Address (IP_Address): 16 data bytes
16 bytes from IP_Address: icmp_seq=0 ttl=113 TOS=0 time=278.813 ms
16 bytes from IP_Address: icmp_seq=1 ttl=113 TOS=0 time=239.935 ms
```

```
16 bytes from IP_Address: icmp_seq=2 ttl=113 TOS=0 time=249.937 ms
16 bytes from IP_Address: icmp_seq=3 ttl=113 TOS=0 time=229.962 ms
16 bytes from IP_Address: icmp_seq=4 ttl=113 TOS=0 time=249.951 ms

--- Host_Address sing statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 229.962/249.720/278.813 ms
[root@godfather bin]#
```

The tcpdump trace:

```
17:28:08.346537 if 4  > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x0]
(ttl 255, id 13170)
                  4508 0024 3372 0000 ff01 083f xxxx xxxx
                  yyyy yyyy 0800 cd8b e704 0000 f8eb bc39
                  8949 0500
17:28:08.625250 if 4  < y.y.y.y > x.x.x.x: icmp: echo reply (ttl 113,
id 12951)
                  4500 0024 3297 0000 7101 9722 yyyy yyyy
                  xxxx xxxx 0000 d58b e704 0000 f8eb bc39
                  8949 0500
```

Oops! Some one zero out my Type-of-Service field!

Before I would let you know who of all Microsoft Windows operating systems did that, I am going to list the Microsoft operating systems who behave correctly – Microsoft Windows 98/SE/ME, Microsoft Windows NT 4 Workstation SP3, Microsoft Windows NT 4 Server SP4, Microsoft Windows NT 4 Workstation SP6a.

The Microsoft Windows 2000 family (Professional, Server, Advanced Server) zero out this field on the ICMP Echo reply.

Is this makes those Microsoft Windows 2000 machines identified easily and uniquely?

99.9% yes. The other 0.01 % belongs to Ultrix & Novel Netware.

From the operating systems I have checked (Linux Kernel 2.2.x, Linux Kernel 2.4 test 2/4/5, FreeBSD 4.0 & 4.1, OpenBSD 2.6 & 2.7, NetBSD 1.4.2, SUN Solaris 2.7 & 2.8, Compaq Tru64 UNIX 5.0, AIX 4.1 & 3.2, OpenVMS v7.2, Irix 6.5.3 & 6.5.8, Ultrix 4.2-4.5, Microsoft Windows 98/SE/ME, Microsoft Windows NT 4 Workstation & Server with various service packs, Microsoft Windows 2000 Professional, Server & Advanced Server) only Ultrix & Novell Netware behaved like the Microsoft Windows 2000 machines.

How can we distinguish between those?
First, there are much fewer Ultrix and Novell operating systems based machines out there than Microsoft's Windows 2000 based machines (I see your faces – not convincing enough).

The fast track in distinguishing between Ultrix, Novell Netware and Microsoft Windows 2000 is a simple one. - By looking at the IP TTL field value within the ICMP Echo replies. Microsoft Windows 2000 family of operating systems use 128 as their default IP TTL field value in ICMP ECHO replies while Ultrix uses 255. The problem is Novell uses the same value for its IP TTL

Field value as Microsoft Windows 2000 based machines (For more information about the IP TTL field value see section 6.3 – „The IP Time-to-Live Field Value with ICMP").

As a next step we can use various types of queries in order to distinguish between the Novell Netware based machines, to the Microsoft Windows 2000 based machines. One of those steps can be an ICMP Timestamp request sent to the "suspicious" machines. No reply from one of the machines will indicate it may be using Novell Netware, a reply from a machine will indicate it is using one of Microsoft Windows 2000 operating system family product. More sophisticated ICMP queries could replace the one I have introduced.

Other ICMP query message types help us to identify a unique group of Microsoft operating systems. As a rule all operating systems except the named Microsoft windows operating systems here, maintain a single behavior regarding the Type-of-Service field. All would maintain the same values with different types of ICMP requests. But, again, Microsoft have some of the "top" people understanding TCP/IP to the degree we humans do not understand.

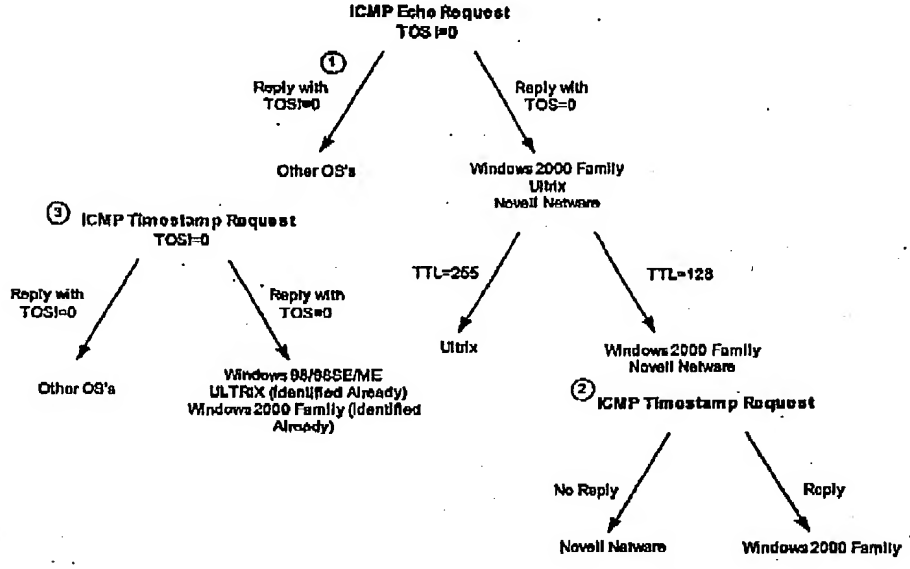

Diagram 7: An example for a way to fingerprint Windows 2000, Ultrix, and Novell Netware based machines with ICMP Query messages with the TOS bits field !=0

We have the following Microsoft operating systems zero out (0x00) the Type-of-Service field with the replies for ICMP Timestamp requests: Microsoft Windows 98/98SE/ME. Microsoft Windows 2000 machines would zero out the TOS field with ICMP Timestamp replies as well.

This means that Microsoft Windows 98/98SE/ME would not zero out the Type-of-Service field value with ICMP Echo requests but will do so with ICMP Timestamp requests.
With the introduced fingerprinting methods in this section we got a way to fingerprint Microsoft Windows 2000, Ultrix, and Novel Netware machines from the rest of the operating systems world. We have a way to distinguish Microsoft Windows 98/98SE/ME (and to set those apart) from the rest of the operating system world, as well.

Operating System	Information Request With TOSI=0x00	Time Stamp Request With TOSI=0x00	Address Mask Request With TOSI=0x00	Echo Request With TOSI=0x00
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	Not Answering	!=0x00	Not Answering	!=0x00
Redhat LINUX 6.2 Kernel 2.2.14	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.0	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 3.4	Not Answering		Not Answering	
OpenBSD 2.7	Not Answering	!=0x00	Not Answering	!=0x00
OpenBSD 2.6	Not Answering	!=0x00	Not Answering	!=0x00
NetBSD	Not Answering	!=0x00	Not Answering	!=0x00
BSDI BSD/OS 4.0	Not Answering	!=0x00	Not Answering	!=0x00
BSDI BSD/OS 3.1	Not Answering	!=0x00	Not Answering	!=0x00
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented			
Solaris 2.7	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.8	Not Implemented	!=0x00	!=0x00	!=0x00
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	!=0x00	!=0x00
Compaq Tru64 v5.0		!=0x00	Not Answering	!=0x00
Irix 6.5.3	Not Answering	!=0x00	Not Answering	!=0x00
Irix 6.5.8	Not Answering	!=0x00	Not Answering	!=0x00
AIX 4.1		!=0x00	Not Answering	!=0x00
AIX 3.2		!=0x00	Not Answering	!=0x00
ULTRIX 4.2 – 4.5		0x00	0x00	0x00
OpenVMS v7.1-2		!=0x00	!=0x00	!=0x00
Novell Netware 5.1 SP1	Not Answering	Not Answering	Not Answering	0x00
Novell Netware 5.0	Not Answering	Not Answering	Not Answering	0x00
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	0x00
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x00	0x00	!=0x00
Windows 98 SE	Not Answering	0x00		!=0x00
Windows ME	Not Answering	0x00	Not Answering	!=0x00
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		!=0x00
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	!=0x00
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	!=0x00
Windows 2000 Professional	Not Answering	0x00	Not Answering	0x00
Windows 2000 Server	Not Answering	0x00	Not Answering	0x00

Table 11: ICMP Query Message Types with TOSI = 0

6.7 Using the TOS Byte's Unused Bit (Fingerprinting Microsoft Windows 2000, ULTRIX and more)

RFC 1349 states that the last field of the TOS byte, the "MBZ" (must be zero), is unused and must be zero. The RFC also states that routers and hosts ignore the value of this bit.

This is the only statement about the unused bit in the TOS Byte in the RFCs. The RFC states: "The originator of a datagram sets this field to Zero".

Obviously it was meant that this field would be always zero. But what will happen if we would set this bit with our ICMP Echo Requests? Will this bit be zero out on reply or will it be echoed back?

Only with ICMP Echo requests we can have a clear identification of OSs.

The next example is an ICMP Echo Request sent with the TOS bit in the TOS Byte set, targeting a FreeBSD 4.1.1 machine:

```
[root@godfather /root]# /usr/local/bin/sing  -c 2 -TOS 1 y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=233 TOS=1 time=330.461 ms
16 bytes from y.y.y.y: seq=1 ttl=233 TOS=1 time=723.300 ms

--- y.y.y.y sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 330.461/526.880/723.300 ms
[root@godfather /root]#
```

Echoing back the Unused bit in the TOS Byte represents the behavior of most of the operating systems I have checked this method against:

Which operating systems are the exceptions?
The next example is with Microsoft Windows 2000 as the targeted machine:

```
[root@godfather precedence_echo]# /usr/local/bin/sing -c 2 -TOS 1
y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=111 TOS=0 time=299.188 ms
16 bytes from y.y.y.y: seq=1 ttl=111 TOS=0 time=280.321 ms

--- y.y.y.y sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 280.321/289.755/299.188 ms
[root@godfather precedence_echo]#
```

The tcpdump trace:

```
00:17:01.765492 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request [tos 0x1]
(ttl 255, id 13170)
                  4501 0024 3372 0000 ff01 d82b xxxx xxxx
                  yyyy yyyy 0800 f015 7a3c 0000 5dc5 0d3a
                  17ae 0b00

00:17:02.064284 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply (ttl 111, id
29961)
                  4500 0024 7509 0000 6f01 2696 yyyy yyyy
                  xxxx xxxx 0000 f815 7a3c 0000 5dc5 0d3a
                  17ae 0b00
```

Another OS that behaves the same is ULTRIX:

```
[root@godfather precedence_echo]# /usr/local/bin/sing -c 2 -TOS 1
y.y.y.y
SINGing to y.y.y.y (y.y.y.y): 16 data bytes
16 bytes from y.y.y.y: seq=0 ttl=237 TOS=0 time=371.776 ms

--- y.y.y.y sing statistics ---
2 packets transmitted, 1 packets received, 50% packet loss
```

```
round-trip min/avg/max = 371.776/371.776/371.776 ms
[root@godfather precedence_echo]#
```

We will use, again, the IP TTL field value to differentiate between the two operating systems.

6.7.1 Changed Pattern with Replies for Different ICMP Query Types

We have a changed pattern with Microsoft Windows 98/98SE/ME when using other ICMP Query message types other than ICMP Echo Request. Instead of echoing this field back, they will zero out this field with their replies.

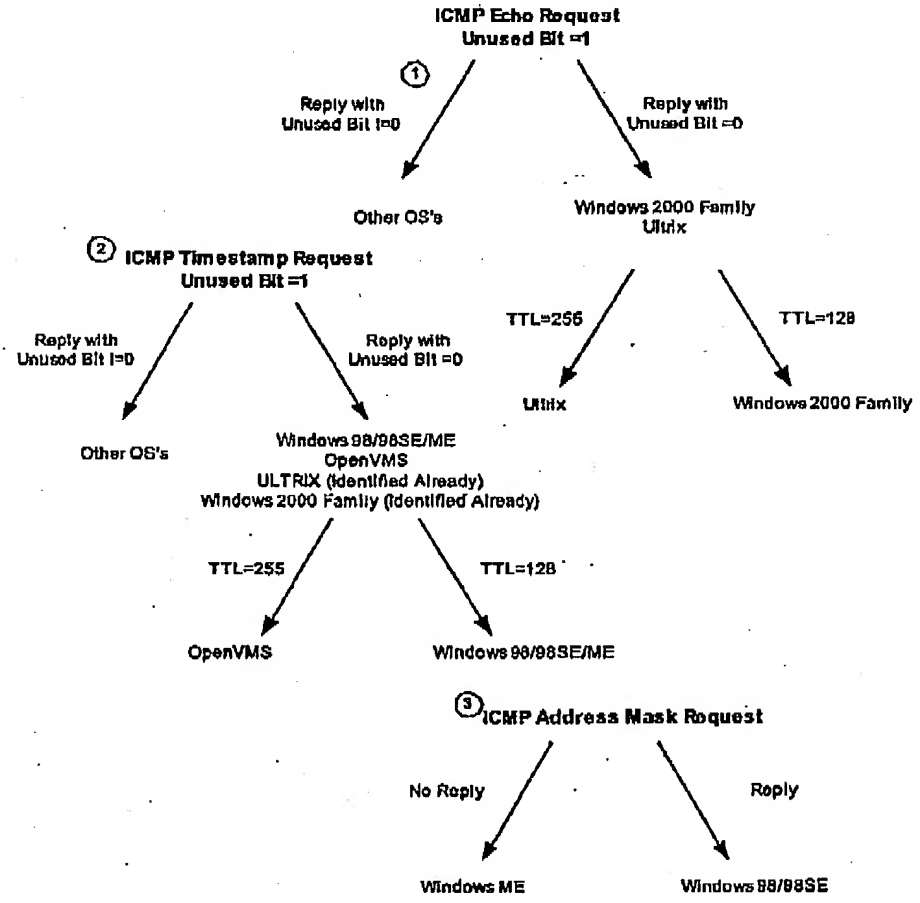

Diagram 8: An example for a way to fingerprint operating systems using the unused bit in the TOS Byte echoing method

Further distinction between the Microsoft operating systems can be achieved if we will query them with ICMP Address Mask request, which only Microsoft Windows 98/98SE will answer for. The Microsoft Windows ME will not reply, enabling us to identify it.

Operating System	Information Request With Unused=1	Time Stamp Request With Unused=1	Address Mask Request With Unused=1	Echo Request With Unused=1
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	Not Answering	0x1	Not Answering	0x1
Redhat LINUX 6.2 Kernel 2.2.14	Not Answering	0x1	Not Answering	0x1
FreeBSD 4.0	Not Answering	0x1	Not Answering	0x1
FreeBSD 4.1.1	Not Answering	0x1	Not Answering	0x1
OpenBSD 2.7	Not Answering		Not Answering	
OpenBSD 2.6	Not Answering		Not Answering	
NetBSD	Not Answering		Not Answering	
BSDI BSD/OS 4.0	Not Answering		Not Answering	
BSDI BSD/OS 3.1	Not Answering		Not Answering	
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented	0x1	0x1	0x1
Solaris 2.7	Not Implemented	0x1	0x1	0x1
Solaris 2.8	Not Implemented	0x1	0x1	0x1
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	0x1	0x1
Compaq Tru64 v5.0	0x1	0x1	Not Answering	0x1
AIX 4.3	0x1	0x1	Not Answering	0x1
AIX 4.2.1	0x1	0x1	Not Answering	0x1
AIX 4.1	0x1	0x1	Not Answering	0x1
AIX 3.2	0x1	0x1	Not Answering	0x1
ULTRIX 4.2 – 4.5	0x0	0x0	0x0	0x0
OpenVMS v7.1-2	0x1	0x1	0x1	0x1
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x0	0x0	0x1
Windows 98 SE	Not Answering	0x0	0x0	0x1
Windows ME	Not Answering	0x0	Not Answering	0x1
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	0x1
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	
Windows 2000 Professional	Not Answering	0x0	Not Answering	0x0
Windows 2000 Server	Not Answering	0x0	Not Answering	0x0

Table 12: ICMP Query Message Types with the TOS Byte Unused Bit value I = 0

6.8 Using the Unused (Identifying Sun Solaris & HP-UX 10.30 & 11.0x OS based machines)

RFC 791 defines a three bits field used for various control flags in the IP Header. Bit 0 of this bits field is the reserved flag, and must be zero according to the RFC.

What will happen if we will decide to break this definition and send our ICMP Query requests with this bit set (having the value of one)?

Sun Solaris & HPUX 11.0x (possibly 10.30 as well) will echo back the reserved bit.

This is a tcpdump trace describing an ICMP Echo request sent with the reserved Bit set, and the ICMP Echo reply we received echoing the reserved bit. This trace was produced against an HP-UX 11.0 machine:

```
21:31:21.033366 if 4  > y.y.y.y > x.x.x.x: icmp: echo request (ttl 255,
id 13170)
                      4500 0024 3372 8000 ff01 fc8c yyyy yyyy
                      xxxx xxxx 0800 8b1b 8603 0000 f924 bd39
                      3082 0000
21:31:21.317916 if 4  < x.x.x.x > y.y.y.y: icmp: echo reply (ttl 236,
id 25606)
                      4500 0024 6406 8000 ec01 def8 xxxx xxxx
                      yyyy yyyy 0000 931b 8603 0000 f924 bd39
                      3082 0000
```

The next trace was produced against a Sun Solaris 2.8 machine:

```
16:51:37.470995 if 4  > 195.72.167.220 > x.x.x.x: icmp: echo request
(ttl 255, id 13170)
                      4500 0024 3372 8000 ff01 e0e1 c348 a7dc
                      xxxx xxxx 0800 edae 3004 0000 69e3 bc39
                      ad2f 0700
16:51:37.745254 if 4  < x.x.x.x > 195.72.167.220: icmp: echo reply (DF)
(ttl 243, id 5485)
                      4500 0024 156d c000 f301 cae6 xxxx xxxx
                      c348 a7dc 0000 f5ae 3004 0000 69e3 bc39
                      ad2f 0700
```

If we examine these traces closely we can identify a distinction between them. The DF bit is set with the Sun Solaris ICMP Query reply and not with the HP-UX 11.0 OS reply.

If you recall from the "DF Bit Playground" section Sun Solaris would set the DF bit by default with all its ICMP Query replies. HP-UX 10.30, and 11.0x operating systems would initiate, by default, a proprietary method in order to determine the PMTU using ICMP Echo requests. After the PMTU is determined the following ICMP Query replies would have the DF bit set in the IP Header.

If we are using only one datagram than in most cases we can distinguish between Sun Solaris and HP-UX 10.30, and 11.0x operating systems since the DF bit will not be set (and if the PMTU is not already determine).

All ICMP Query replies on the same operating system use the same pattern (either echo the reserved bit with all replies or not). This enable us to use another ICMP Query message type for this fingerprinting method. If we send an ICMP Address Mask request with the reserved bit set, the result a Sun Solaris 2.8 machine will produce will be something like the next trace:

```
18:39:32.262869 if 4  > y.y.y.y > x.x.x.x : icmp: address mask request
(ttl 255, id 13170)
                    4500 0020 3372 8000 ff01 e12e yyyy yyyy
                    xxxx xxxx 1100 a0fb 4e04 0000 0000 0000
18:39:32.561373 if 4  < x.x.x.x > y.y.y.y: icmp: address mask is
0xffffff00 (DF) (ttl 243, id 51792)
                    4500 0020 ca50 c000 f301 1650 xxxx xxxx
                    yyyy yyyy 1200 a0fa 4e04 0000 ffff ff00
```

We will have both the reserved bit and the DF bit set on the ICMP Address Mask reply, a unique pattern Sun Solaris machines have with ICMP Query replies.

This operating system fingerprinting method enables us to identify and distinguish between Sun Solaris, and HP-UX 10.30 &11.0x operating systems to the other operating systems.

I have asked Alfredo Andres Omella, author of SING, to incorporate the ability to set the reserved bit with his tool. Alfredo has introduced the –U option along with the ability to identify if this bit is set on the reply (if any) we get, with the latest version of SING:

```
[root@godfather bin]# ./sing -mask -U IP_Address
SINGing to IP_Address (IP_Address): 12 data bytes
12 bytes from IP_Address: icmp_seq=0 RF! DF! ttl=243 TOS=0 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=1 RF! DF! ttl=243 TOS=0 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=2 RF! DF! ttl=243 TOS=0 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=3 RF! DF! ttl=243 TOS=0 mask=255.255.255.0
12 bytes from IP_Address: icmp_seq=4 RF! DF! ttl=243 TOS=0 mask=255.255.255.0
— IP_Address sing statistics —
5 packets transmitted, 5 packets received, 0% packet loss
[root@godfather bin]#
```

This method was tested against: Linux Kernel 2.4 test 2,4,5,6; Linux Kernel 2.2.x; FreeBSD 4.0, 3.4; OpenBSD 2.7,2.6; NetBSD 1.4.1,1.4.2; BSDI BSD/OS 4.0,3.1; Solaris 2.6,2.7,2.8; HP-UX 10.20, 11.0; Compaq Tru64 5.0; Aix 4.1,3.2; Irix 6.5.3, 6.5.8; Ultrix 4.2 – 4.5; OpenVMS v7.1-2; Novel Netware 5.1 SP1, 5.0, 3.12; Microsoft Windows 98/98SE, Microsoft Windows NT WRKS SP6a, Microsoft Windows NT Server SP4, Microsoft Windows 2000 Family.

6.9 DF Bit Echoing

Some operating systems, when receiving an ICMP Query message with the DF bit set, would set the DF bit with their replies as well. Sometimes it would be in contrast with their regular behavior, which would be not setting the DF Bit in their replies for a regular query that comes with the DF bit not set.

6.9.1 DF Bit Echoing with the ICMP Echo request

The tcpdump trace below illustrates an ICMP Echo request sent from a Linux box, using SING[42], to a SUN Solaris 2.7 machine:

```
[root@godfather bin]# ./sing -echo -G IP_Address
SINGing to IP_Address (IP_Address): 16 data bytes
16 bytes from IP_Address: icmp_seq=0 DF! ttl=243 TOS=0 time=188.289 ms
16 bytes from IP_Address: icmp_seq=1 DF! ttl=243 TOS=0 time=250.026 ms
16 bytes from IP_Address: icmp_seq=2 DF! ttl=243 TOS=0 time=240.298 ms
16 bytes from IP_Address: icmp_seq=3 DF! ttl=243 TOS=0 time=260.036 ms

--- IP_Address sing statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 188.289/234.662/260.036 ms
[root@godfather bin]#
```

The tcpdump trace:

```
17:16:23.527050 if 4 > x.x.x.x > y.y.y.y: icmp: echo request (DF) (ttl
255, id 13170)
                4500 0024 3372 4000 ff01 20f0 xxxx xxxx
```

[42] The –G option with SING sets the DF Bit with the ICMP Query requests.

```
                    yyyy yyyy 0800 a5cd b304 0000 37e9 bc39
                    a30a 0800
17:16:23.715250 if 4  < y.y.y.y > x.x.x.x: icmp: echo reply (DF) (ttl
243, id 18227)
                    4500 0024 4733 4000 f301 192f yyyy yyyy
                    xxxx xxxx 0000 adcd b304 0000 37e9 bc39
                    a30a 0800
```

Most of the operating systems that I have checked this behavior against did the same thing. In the reply they produced, the DF bit was set.

Which operating systems are the exceptional and do not echo back the DF bit?
Linux operating systems based on Kernel 2.2.x, and Kernel 2.4 with the various test kernels, Ultrix v4.2 – 4.5, and Novell Netware.

How can we distinguish between those operating systems?
Frankly it is quite simple. Since LINUX and Ultrix are using an IP TTL field value of 255 in their ICMP Query replies, and Novell Netware uses 128, it is easy to distinguish between those groups. If we want to further distinguish between LINUX based systems and Ultrix based systems, we can send an ICMP Information request or an ICMP Address Mask request to the questioned machines. The machine, which would answer those, will be the one based on the Ultrix operating system.

6.9.2 DF Bit Echoing with the ICMP Address Mask request

With ICMP Address Mask requests we have a different story. Among the operating systems that I have checked that answer for an ICMP Address Mask request Sun Solaris & OpenVMS echo back the DF bit. Microsoft Windows 98, Microsoft Windows 98 SE, and Ultrix do not echo back the DF bit.

Again it is very simple to distinguish between the Microsoft Windows 98 family and between the Ultrix machines. Since the Microsoft Windows 98 family is using 128 as their IP TTL field value in their ICMP query replies and Ultrix uses 255, we can distinguish between those operating systems.

We have here a simple method to distinguish between Microsoft Windows 98 / 98 SE, and Ultrix machines to the rest of the operating systems world.

Another interesting piece of information is that the Microsoft Windows 98 family changed its behavior from DF echoing with the ICMP Echo request to not echoing with the ICMP Address Mask request. This inconsistency is a factor with all Microsoft operating systems (Echoing with ICMP Echo request, not echoing with the other types of ICMP query).

6.9.3 DF Bit Echoing with the ICMP Timestamp request

Since a lot more operating systems answer for an ICMP Timestamp request than with the ICMP Address Mask request, we have a bit more difficulty in identifying those.

Linux machines based on Kernel 2.2.x, or Kernel 2.4, Ultrix, Microsoft Windows 98/98SE/ME, and the Microsoft Windows 2000 Family would not echo back the DF bit with ICMP Timestamp replies they produce for ICMP Timestamp request that sets their DF bit.

Here we can only distinguish between certain groups of operating systems; again it would be according to their IP TTL field value with their replies.

Linux would use 255 as its TTL field value for the ICMP Timestamp reply; Ultrix would use the same value. The Microsoft family of operating systems that would answer for this kind of query would use 128 as their TTL value.

Again we have Linux and Ultrix on the one hand and the Microsoft Family on the other hand. How can we further distinguish between those? We can correlate the information (as discussed in the next section, or query the "suspicious" machines with another type of ICMP Query message).

6.9.4 Using all of the information in order to identify maximum of operating systems
We can group Linux and Ultrix with the ICMP Echo requests. We can do the same with Microsoft Windows 98 / 98 SE & Ultrix using the ICMP Address Mask requests. This would allow us to pinpoint the Linux boxes from the first stage. So when we would go into the third stage we would know which operating systems are Linux based, which are Microsoft Windows 98 / 98 SE based, and which are Ultrix based. This would leave us with Microsoft Windows ME and with the Microsoft Windows 2000 family machines.

6.9.5 Why this would work (for the skeptical)
All those skeptical would say that if they receive an ICMP Query request with the DF bit set than it should be clear that something is wrong and someone is probably trying to scan them. Think again: What would happen if a Sun Solaris machine will query your machine? Than the same behavior would be produced.

This is an ICMP Echo request sent from a Solaris 2.6 box to a Linux box. We can see that the DF bit is set with the request and not set with the reply. But again if some one would mimic this behavior with a tool used on a Linux box to query the world, which is 100% mimicking a Sun Solaris request than we would never know if this is a legit request or an attempt for scanning / fingerprinting.

```
Initializing Network Interface...
Decoding raw data on interface ppp0

-*> Snort! <*-
Version 1.6
By Martin Roesch (roesch@clark.net, www.clark.net/~roesch)
08/10-23:32:52.201612 y.y.y.y -> 139.92.207.58
ICMP TTL:239 TOS:0x0 ID:48656  DF
ID:2080   Seq:0  ECHO
39 93 10 A3 00 03 F0 E5 08 09 0A 0B 0C 0D 0E 0F  9...............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567


08/10-23:32:52.201649 139.92.207.58 -> Y.Y.Y.Y
ICMP TTL:255 TOS:0x0 ID:349
ID:2080   Seq:0  ECHO REPLY
39 93 10 A3 00 03 F0 E5 08 09 0A 0B 0C 0D 0E 0F  9...............
10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F  ................
20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F   !"#$%&'()*+,-./
30 31 32 33 34 35 36 37                          01234567
```

Operating System	Info. Request	Time Stamp Request	Address Mask Request	Echo Request
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	Not Answering	+ (- DF)	Not Answering	+ (- DF)
Redhat LINUX 6.2 Kernel 2.2.14	Not Answering	+ (- DF)	Not Answering	+ (- DF)
FreeBSD 4.0	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
FreeBSD 3.4	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
OpenBSD 2.7	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
OpenBSD 2.6	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
NetBSD	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
BSDI BSD/OS 4.0	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
BSDI BSD/OS 3.1	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Solaris 2.5.1	Not Answering			
Solaris 2.6	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
Solaris 2.7	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
Solaris 2.8	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	+ (+ DF)	+ (+ DF)
Compaq Tru64 v5.0		+ (+ DF)	Not Answering -	+ (+ DF)
Irix 6.5.3	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Irix 6.5.8	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
AIX 4.1		+ (+ DF)	Not Answering	+ (+ DF)
AIX 3.2		+ (+ DF)	Not Answering	+ (+ DF)
ULTRIX 4.2 – 4.5		+ (- DF)	+ (- DF)	+ (- DF)
OpenVMS v7.1-2		+ (+ DF)	+ (+ DF)	+ (+ DF)
Novell Netware 5.1 SP1	Not Answering	Not Answering	Not Answering	+ (- DF)
Novell Netware 5.0	Not Answering	Not Answering	Not Answering	+ (- DF)
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	+ (- DF)
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	+ (- DF)	+ (- DF)	+ (+ DF)
Windows 98 SE	Not Answering	+ (- DF)	+ (- DF)	+ (+ DF)
Windows ME	Not Answering	+ (- DF)	Not Answering	+ (+ DF)
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	+ (+ DF)
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	+ (+ DF)
Windows 2000 Professional	Not Answering	+ (- DF)	Not Answering	+ (+ DF)
Windows 2000 Server	Not Answering	+ (- DF)	Not Answering	+ (+ DF)

Table 13: DF Bit Echoing

6.9.6 Combining all together

If we combine every thing together than we can start from sending ICMP Echo requests with the DF bit set probing the targeted host(s) / network(s). The operating systems, which will not echo the DF bit with their ICMP Query replies, will be Linux operating system machines based either on kernel 2.2.x, or on Kernel 2.4.x, Novell Netware, and Ultrix machines. We can distinguish between the Novell Netware machines, the Linux based machines, and the Ultrix based machines according to the IP TTL field values with the ICMP Echo replies.

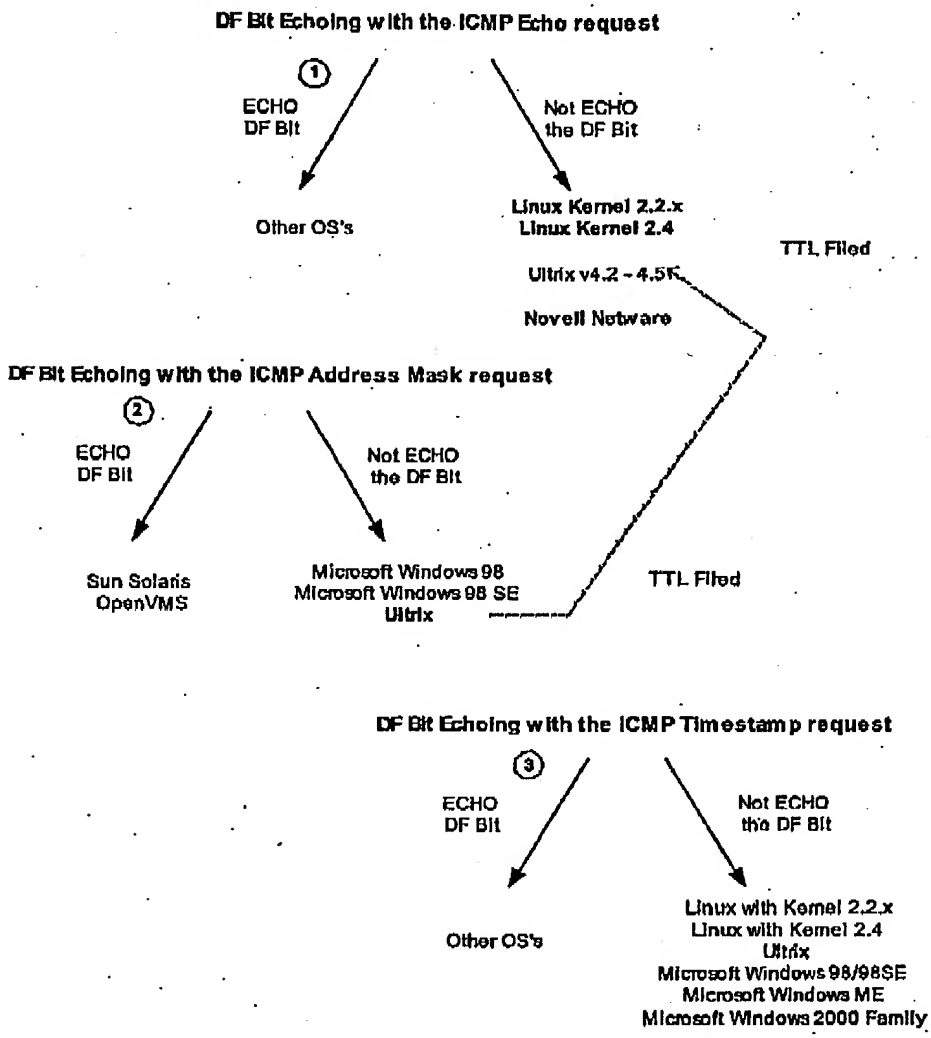

Diagram 9: An example of fingerprinting using the DF Bit Echoing technique

The second stage would be sending ICMP Address Mask requests with the DF bit set to the same targeted host(s). Microsoft Windows 98/98 SE and Ultrix based machines would not echo the DF bit with their ICMP Query reply. We than can distinguish between the Ultrix machines and the Microsoft Windows machines, because of the different IP TTL field values in the ICMP Address Mask replies.

We can now also identify the Ultrix machines from the first step – we know their IPs now. Than it leaves us with only the Linux boxes. Within two steps we are able of fingerprinting Novell Netware, Ultrix, Microsoft Windows 98/98 SE and Linux operating systems based on kernel 2.2.x or on kernel 2.4.x.

In the last step of this example we are sending ICMP Timestamp requests with the DF bit set to the same group of IPs we are probing. The operating systems which do not echo back the DF bit in their ICMP Query replies are Linux operating systems based on Kernel 2.2.x, or on Kernel 2.4, Ultrix, Microsoft Windows 98/98SE, Microsoft Windows ME, and Microsoft Windows 2000 Family. Since we already fingerprinted most of the operating systems in this it enable us to fingerprint Microsoft Windows ME, and Microsoft Windows 2000 family based machines.

6.10 Using Code field values different than zero within ICMP ECHO requests

An interesting detail I have discovered during the lab experiments I did when I have researched ICMP scanning is when a wrong code is sent along with the correct type of ICMP query message, different operating systems would send different code values back.

In the next example I have sent an ICMP Echo Request with the code field value set to 38 instead of 0, to a LINUX machine running Redhat LINUX 6.2 Kernel 2.2.14.

We can look at the tcpdump trace, the type and code fields are in bold type:

```
00:21:05.238649 ppp0 > x.x.x.x > y.y.y.y: icmp: echo request (ttl 255,
id 13170)
                    4500 0024 3372 0000 ff01 08d3 xxxx xxxx
                    yyyy yyyy 0826 af13 2904 0000 41e4 c339
                    17a4 0300
00:21:05.485617 ppp0 < y.y.y.y > x.x.x.x: icmp: echo reply (ttl 240, id
2322)
                    4500 0024 0912 0000 f001 4233 yyyy yyyy
                    xxxx xxxx 0026 b713 2904 0000 41e4 c339
                    17a4 0300
```

In the ICMP Echo reply LINUX produced the code field value is set to 38.

If we examine what RFC 792 requires, we see that LINUX does exactly that.

The sending side initializes the identifier (used to identify ECHO requests aimed at different destination hosts) and sequence number (if multiple ECHO requests are sent to the same destination host), adds some data (arbitrary) to the data field and sends the ICMP ECHO Request to the destination host. *In the ICMP header the code equals zero.* The recipient should *only change* the type to ECHO Reply and return the datagram to the sender.

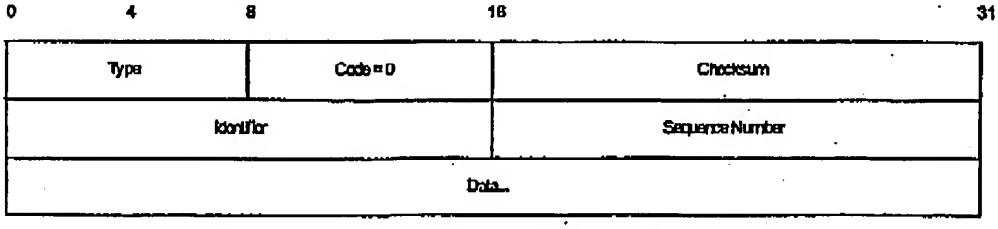

Figure 12: ICMP ECHO Request & Reply message format

This also means that we trust another machine to behave correctly, when that host produce the ICMP Echo reply.

LINUX changes the type field value to 0 and sends the reply. The code field is unchanged.

I have checked the behavior of my Microsoft Windows 2000 Professional box. I have sent the same ICMP ECHO Request message to the Microsoft Windows box (the code field is in bold type):

```
10:03:33.860212 eth0 > localhost.localdomain > 192.168.1.1: icmp: echo
request
                          4500 0020 3372 0000 fe01 0614 c0a8 0105
                          c0a8 0101 0826 d618 6102 f658 0183 c8e2

10:03:33.860689 eth0 < 192.168.1.1 > localhost.localdomain: icmp: echo
reply
                          4500 0020 2010 0000 8001 9776 c0a8 0101
                          c0a8 0105 0000 de3e 6102 f658 0183 c8e2
                          0000 0000 0000 0000 0000 0000 0000
```

The Microsoft Windows 2000 Professional operating system changed the code field value on the ICMP Echo Reply to the value of 0.

This method was tested with various operating systems including LINUX Kernel 2.4t1-8, IBM AIX 4.x & 3.2, SUN Solaris 2.51, 2.6, 2.7 & 2.8, OpenBSD 2.6 & 2.7, NetBSD 1.4.1, 1.4.2, BSDI BSD/OS 4.0 & 3.1, HP-UX 10.20 & 11.0, Compaq Tru64 v5.0, Irix 6.5.3 & 6.5.8, Ultrix 4.2-4.5, OpenVMS, FreeBSD 3.4, 4.0 & 4.1 and they produced the same results as the LINUX box (Kernel 2.2.x) did.

Microsoft Windows 4.0 Server SP4, Microsoft Windows NT 4.0 Workstation SP 6a, Microsoft Windows NT 4.0 Workstation SP3, Microsoft Windows 95 / 98 / 98 SE / ME have produced the same behavior as the Microsoft Windows 2000 Professional (Server & Advanced Server).

We have a fingerprinting method to differentiate between a Microsoft Windows based machine to the rest of the operating systems world using code values, which are different than zero, inside ICMP Echo Requests.

6.11 Using Code field values different than zero within ICMP Timestamp Request

I have decided to map which operating systems would answer to an ICMP Timestamp Request that would have its code field not set to zero, and how the ICMP Timestamp reply (if any) will help us identify those operating systems.

6.11.1 The non-answering Operating Systems

Interesting results were produced. The Microsoft Windows 98/98 SE/ME, and the Microsoft Windows 2000 Family that have answered to ICMP Timestamp requests with the code field set to zero, now did not produce any reply back.

This enables us to group together certain versions of the Windows Operating System.

The next diagram shows how we can distinguish between the different Microsoft Windows operating systems using two datagrams of ICMP Timestamp request. The first one is a regular one; the Microsoft Windows machines that do not answer are Microsoft Windows 95 and Microsoft Windows NT 4.0 Workstation with SP 6a. All other operating systems (that I have tested) answer the ICMP Time stamp request. The second stage is sending another datagram, this time with the Code field set to a value, which is not equal to zero. The operating systems that would not answer will include Windows 98/98 SE/ME/2000 Family, which are the newer versions of Microsoft Windows operating systems.

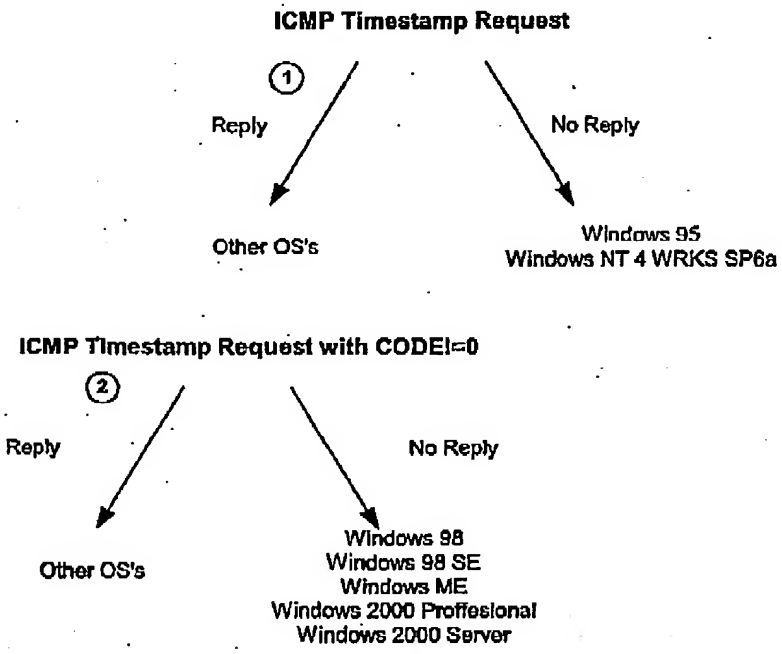

Diagram 10: Finger Printing Using ICMP Timestamp Request and Wrong Codes

It is quite obvious that Microsoft have tried to change some of their newer operating systems fingerprinting in later TCP/IP implementations of their operating systems. For example, the default for answering an ICMP Timestamp request was changed from "no answer" to "answer", like UNIX and UNIX-like operating systems. But the Microsoft programmers / designers / architects / security engineers did not think about every thing apparently.

6.11.2 Operating Systems the Zero out the Code field value on Reply

I was looking to see if there are any operating systems in which answered the crafted ICMP Timestamp Query with the Code field set to a value different than zero, which might zero out this field value in its ICMP Echo Reply.

I have found that LINUX operating systems based on Kernel 2.2.x or on the 2.4 Kernel (with the various test Kernels) zero out the code field with the ICMP Echo replies they produce. The next trace is a tcpdump trace describing ICMP Echo Request and reply from a LINUX 2.4 test Kernel 6, to a crafted ICMP Echo Request with a code field different than zero:

```
[root@godfather /root]# sing  -tstamp -x 38 -c 2 IP_Address
SINGing to IP_Address (IP_Address): 20 data bytes
20 bytes from IP_Address: icmp_seq=0 ttl=243 TOS=0 diff=24315927
20 bytes from IP_Address: icmp_seq=1 ttl=243 TOS=0 diff=24316176

--- IP_Address sing statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
[root@godfather /root]#


20:10:18.138486 ppp0 > x.x.x.x > y.y.y.y: icmp: time stamp request (ttl
255, id 13170)
                  4500 0028 3372 0000 ff01 606c xxxx xxxx
                  yyyy yyyy 0d26 2e0c 7c04 0000 03af 451a
                  0000 0000 0000 0000
20:10:18.354222 ppp0 < y.y.y.y > x.x.x.x: icmp: time stamp reply (ttl
243, id 15717)
                  4500 0028 3d65 0000 f301 6279 yyyy yyyy
                  xxxx xxxx 0e00 888b 7c04 0000 03af 451a
                  0422 4e31 0422 4e31
20:10:19.134165 ppp0 > x.x.x.x > y.y.y.y: icmp: time stamp request (ttl
255, id 13170)
                  4500 0028 3372 0000 ff01 606c xxxx xxxx
                  yyyy yyyy 0d26 2928 7c04 0100 03af 48fe
                  0000 0000 0000 0000
20:10:19.354210 ppp0 < y.y.y.y > x.x.x.x: icmp: time stamp reply (ttl
243, id 15718)
                  4500 0028 3d66 0000 f301 6278 yyyy yyyy
                  xxxx xxxx 0e00 7bed 7c04 0100 03af 48fe
                  0422 520e 0422 520e
```

6.11.3 Changed Patterns

The LINUX operating system behavior with the crafted ICMP Timestamp requests is in contrast with its behavior with the crafted ICMP Echo Requests sent with the Code field set to a value different than zero.

This also gives us a unique piece of information that enables us to identify LINUX machines better.

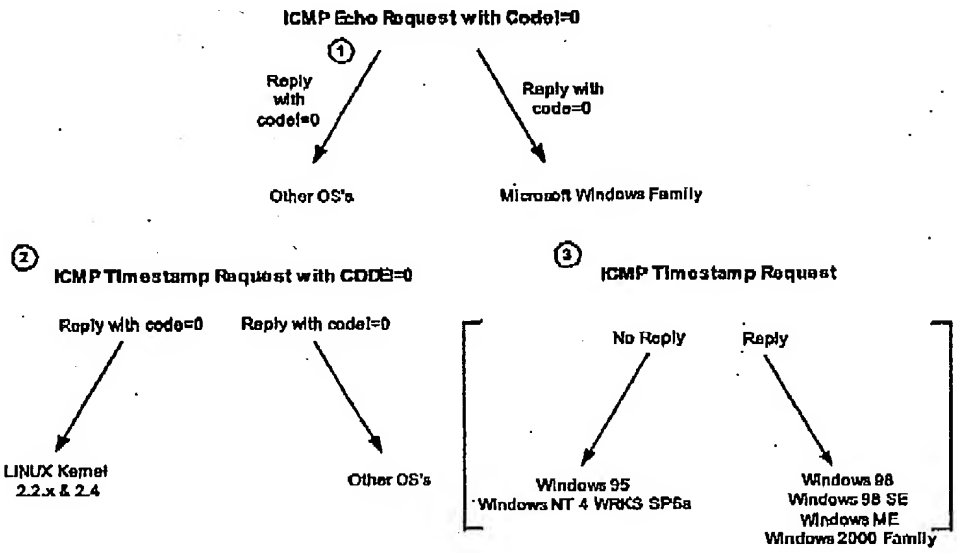

Diagram 11: An Example of Finger Printing Using crafted ICMP Echo & Timestamp Request

The diagram above describes a process in which we can use in order to differentiate between certain groups of operating systems.

The first step is sending an ICMP Echo request with the code field set to a value different than zero. The ICMP Echo replies with the code field equal to zero would distinguish the Microsoft based operating systems group, from the other UNIX and UNIX-like operating systems.

Sending ICMP Timestamp requests with the Code field value different than zero to all participants of the group of the UNIX and UNIX-like operating systems will identify LINUX 2.2.x and 2.4 Kernel based machines (since they zero out the code field in their replies).

Sending ICMP Timestamp request to the Microsoft Windows based group of operating systems will separate the group to those machines rather being windows 95 or windows NT 4 SP4 and above (not answer the query), to those that may be one of the following – Microsoft Windows 98 / SE / ME / Windows 2000 Family (answer the query).

For a list of ICMP Query message types sent to different types of operating systems with the code field set to a value different than zero, and the various ICMP Query replies we got back (if any) please see "Appendix D: ICMP Query Message types with Code field !=0 (table)".

Using the ICMP Error Messages

6.12 Operating system, which do not generate ICMP Protocol Unreachable Error Messages

Several operating systems will not generate an ICMP Protocol Unreachable error message, when one is expected to be produced, in response to an offending datagram trying to use a protocol, which is not used on those operating systems.

Those operating systems include:

- AIX
- DG-UX
- HP-UX

6.13 ICMP Error Message Quenching

RFC 1812 and RFC 1122 suggest limiting the rate at which various error messages are sent. Only few operating systems are known to follow this.

An attacker can use this to send UDP packets to a random, high UDP port and count the number of ICMP Destination unreachable messages received within a given amount of time.

6.14 ICMP Error Message Quoting Size

Each ICMP error message includes the Internet Protocol (IP) Header and *at least* the first 8 data bytes of the datagram that triggered the error (the offending datagram); more than 8 bytes *may* be sent according to RFC 1122.

Most of the operating systems will quote the offending packets IP Header and the first 8 data bytes of the datagram that triggered the error. Several operating systems and networking devices will parse the RFC guidelines a bit different and will echo more than 8 bytes.

Which operating systems will quote more?
LINUX based on Kernel 2.0.x/2.2.x/2.4.t-x, Sun Solaris, HPUX 11.x, MacOS 7.55/8.x/9.04, Nokia boxes, Foundry Switches (and other OSs and several Networking Devices) are a good example.

The fact is not new. Fyodor outlined this in his article "Remote OS Identification by TCP/IP Fingerprinting"[43].

The idea is in trying to differentiate between the different operating systems that quote more than the usual. How can this be done? Looking for example on the amount of information quoted. Is there a limit to the quoted size? Will the quoted data be the entire offending packet or just part of it? Will the quoted data be the echoed correctly? Will extra bytes will be padded to the echoed data? and some other parameters.

The next example is with Sun Solaris 7. I have sent a UDP datagram to a closed UDP port:

```
00:13:35.559947 ppp0 > x.x.x.x.1084 > y.y.y.y.2000: udp 0 (ttl 64, id
44551)
                  4500 001c ae07 0000 4011 7aa4 xxxx xxxx
                  yyyy yyyy 043c 07d0 0008 a1ac

00:13:35.923691 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 2000
unreachable Offending pkt: x.x.x.x.1084 > y.y.y.y.2000: udp 0 (ttl 45,
id 44551) (DF) (ttl 236, id 63417)
                  4500 0038 f7b9 4000 ec01 44e5 yyyy yyyy
                  xxxx xxxx 0303 4f3c 0000 0000 4500 001c
                  ae07 0000 2d11 8da4 xxxx xxxx yyyy yyyy
```

[43] http://www.insecure.org/nmap/nmap-fingerprinting-article.html

```
                    043c 07d0 0008 a1ac
```

Please note that for having more than 8 data bytes quoted, you need to have data in the offending datagram. If not, there is nothing to quote beyond the regular 8 bytes (usually, if the OS is not padding other data bytes).

The next example is with Sun Solaris 8. I have sent a UDP datagram to a closed UDP port, adding 80 bytes of data to the datagram.

```
[root@godfather]# hping2 -2 -d 80 -c 1 y.y.y.y
eth0 default routing interface selected (according to /proc)
HPING y.y.y.y (eth0 y.y.y.y): udp mode set, 28 headers + 80 data bytes
ICMP Port Unreachable from y.y.y.y (y.y.y.y)

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

The tcpdump trace:

```
11:52:50.830383 eth0 > x.x.x.x.2198 > y.y.y.y.0: udp 80 (ttl 64, id
17240)
                4500 006c 4358 0000 4011 99ae xxxx xxxx
                yyyy yyyy 0896 0000 0058 8b5f 5858 5858
                5858 5858 5858 5858 5858 5858 5858 5858
                5858 5858 5858 5858 5858 5858 5858 5858
                5858 5858 5858 5858 5858 5858 5858 5858
                5858 5858 5858 5858 5858 5858 5858 5858
                5858 5858 5858 5858 5858 5858

11:52:51.367331 eth0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2198 > y.y.y.y.0: udp 80 (ttl 48, id
17240) (DF) (ttl 231, id 49576)
                4500 0070 c1a8 4000 e701 3469 yyyy yyyy
                xxxx xxxx 0303 bf05 0000 0000 4500 006c
                4358 0000 3011 a9ae xxxx xxxx yyyy yyyy
                0896 0000 0058 8b5f 5858 5858 5858 5858
                5858 5858 5858 5858 5858 5858 5858 5858
                5858 5858 5858 5858 5858 5858 5858 5858
                5858 5858 5858 5858 5858 5858 5858 5858
```

The result is an ICMP Port Unreachable Error message that will echo only 64 bytes of the offending datagram's data portion.

The limit of 64 bytes quoted from the offending packet's data portion is not limited to Sun Solaris only. HPUX 11.x, MacOS 7.55/8.x/9.04, will do the same.

Other operating systems / networking devices will have their own barriers. For example, LINUX based on Kernel 2.2.x/2.4.x-t will send and ICMP Error Message up to 576 bytes long. LINUX will quote 528 bytes from the data portion of the offending packet (576 minus 20 bytes of usuall IP Header, minus 8 bytes of the ICMP Header, minus the offending packet's IP Header that is 20 bytes will leave you with 528 bytes of data portion. This is no IP options are presented).

I know an operating system, and a family of networking devices that will pad extra data to the echoed offending packet. LINUX case is detailed in the next section. The next example is with

Foundry Networks ServerIron running software version 07.1.02T12. I have sent a UDP datagram to a closed UDP port on the Foundry switch:

```
[root@godfather]# hping2 -2 -c 1 y.y.y.y
eth0 default routing interface selected (according to /proc)
HPING y.y.y.y (eth0 y.y.y.y): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from y.y.y.y (y.y.y.y)

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather]#


12:08:47.793503 eth0 > x.x.x.x.2498 > y.y.y.y.0: udp 0 (ttl 64, id
44437)
                  4500 001c ad95 0000 4011 885f xxxx xxxx
                  yyyy yyyy 09c2 0000 0008 b13f

12:08:48.240208 eth0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2498 > y.y.y.y.0: udp 0 (ttl 51, id
44437) (ttl 51, id 17453)
                  4500 0044 442d 0000 3301 feaf yyyy yyyy
                  xxxx xxxx 0303 739c 0000 0000 4500 001c
                  ad95 0000 3311 955f xxxx xxxx yyyy yyyy
                  09c2 0000 0008 b13f dd2c 2a16 38a1 7646
                  7aaa 9d41
```

As it seems Foundry switches will pad 12 bytes with ICMP Port unreachable.

Other fingerptinting facts that are outlined through this section will halp us to differentiate between the operating systems, which carry the same behavior.

I have examined three ICMP Error Messages a Host can issue:

- ICMP Port Unreachable
- ICMP Protocol Unreachable
- ICMP IP Reassembly Time Exceeded

Other ICMP Error Messages, which a Host can issue and should be checked to see if they hold more fingerprinting differences, are:

- Source Quench
- Parameter Problem

6.15 LINUX ICMP Error Message Quoting Size Differences / The 20 Bytes from No Where

We must understand that there are differences between the different ICMP Error messages, not only with their meaning, but also with their implementation. I was expecting that several characters with the ICMP Error messages will be the same along all of the ICMP Error Messages, but I was wrong regarding few operating systems.

The most interesting case is with the LINUX operating system based on Kernel 2.2.x and 2.4.t-x.

The next example is with LINUX based on Kernel 2.2.16 as the targeted machine, eliciting an ICMP Port Unreachable error message:

```
00:21:30.199408 ppp0 > x.x.x.x.2066 > y.y.y.y.2000: udp 0 (ttl 64, id
1732)
                    4500 001c 06c4 0000 4011 c895 xxxx xxxx
                    yyyy yyyy 0812 07d0 0008 4484

00:21:30.493691 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 2000
unreachable Offending pkt: x.x.x.x.2066 > y.y.y.y.2000: udp 0 (ttl 44,
id 1732) [tos 0xc0]  (ttl 238, id 53804)
                    45c0 0038 d22c 0000 ee01 4e60 yyyy yyyy
                    xxxx xxxx 0303 a88e 0000 0000 4500 001c
                    06c4 0000 2c11 dc95 xxxx xxxx yyyy yyyy
                    0812 07d0 0008 4484
```

The quoted data is the entire offending datagram. LINUX ICMP Error messages will be up to 576 bytes long according to the LINUX source code.

The next example is with LINUX as the targeted operating system. With this example I have sent a protocol scan with NMAP:

```
13:14:56.942897   < x.x.x.x > y.y.y.y: ip-proto-38 0 (ttl 39, id 37623)
                    4500 0014 92f7 0000 2726 02cb xxxx xxxx
                    yyyy yyyy
13:14:56.942964   > y.y.y.y > x.x.x.x: icmp: y.y.y.y protocol 38
unreachable Offending pkt: x.x.x.x > y.y.y.y: ip-proto-38 0 (ttl 39, id
37623) [tos 0xc0]  (ttl 255, id 1884)
                    45c0 0044 075c 0000 ff01 b59a yyyy yyyy
                    xxxx xxxx 0302 fb1a 0000 0000 4500 0014
                    92f7 0000 2726 02cb xxxx xxxx yyyy yyyy
                    0050 dc84 ae6f 6910 0000 0000 5004 0000
                    bd89 0000
```

LINUX adds to the entire offending packet that was quoted, another 20 bytes.

Since LINUX handles the ICMP Protocol Unreachable Error Messages like the ICMP Fragment Reassembly Time Exceeded Error Messages we will see the same pattern with ICMP Fragment Reassembly Time Exceeded:

```
[root@godfather bin]# hping2 -c 1 -x -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y ppp0 y.y.y.y): NO FLAGS are set, 40 headers + 0 data
bytes

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather bin]#
```

The tcpdump trace:

```
19:49:22.999108 ppp0 > x.x.x.x.cvspserver > y.y.y.y.0: .
1709055398:1709055398(0) win 512 (frag 35247:20@0+) (DF) (ttl 64)
                    4500 0028 89af 6000 4006 e0ff xxxx xxxx
                    yyyy yyyy 0961 0000 65de 1da6 6a01 476b
                    5000 0200 bf71 0000

19:49:53.303196 ppp0 < y.y.y.y > x.x.x.x: icmp: ip reassembly time
exceeded Offending pkt: x.x.x.x.cvspserver > y.y.y.y.0: .
1709055398:1709055398(0) win 512 (frag 35247:20@0+) (DF) (ttl 45) [tos
0xc0]  (ttl 238, id 379)
                    45c0 0058 017b 0000 ee01 1a49 yyyy yyyy
                    xxxx xxxx 0b01 3caf 0000 0000 4500 0028
                    89af 6000 2d06 f3ff xxxx xxxx yyyy yyyy
                    0961 0000 65de 1da6 6a01 476b 5000 0200
                    bf71 0000 601d 1f0d 7a04 5045 0100 0000
                    4146 4345 4a45 4f46
```

Since LINUX's ICMP Error messages will not be bigger than 576 bytes long, if the offending packet will be big enough (not likely in real world situation) we will not see the added 20 bytes in the ICMP Fragment Reassembly / ICMP Protocol Unreachable error messages.

This unique pattern will allow us to identify LINUX based machines even if the Precedence Bits value with the LINUX ICMP Error messages will be changed to 0x000.

6.16 Foundry Networks Networking Devices Padded Bytes with ICMP Port Unreachable(s) / The 12 Bytes from No Where

Linux is not the only operating system that will have weird data bytes padded to one of his ICMP Error messages.

Foundry Network's networking devices will pad extra 12 bytes of data with their ICMP Port Unreachable Error messages. Our first example is with a ServerIron switch running software version 7.1.02T12, eliciting an ICMP Port Unreachable error message:

```
[root@godfather]# hping2 -2 -c 1 y.y.y.y
eth0 default routing interface selected (according to /proc)
HPING y.y.y.y (eth0 y.y.y.y): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from y.y.y.y (y.y.y.y)

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather]#


12:08:47.793503 eth0 > x.x.x.x.2498 > y.y.y.y.0: udp 0 (ttl 64, id
44437)
                    4500 001c ad95 0000 4011 885f xxxx xxxx
                    yyyy yyyy 09c2 0000 0008 b13f

12:08:48.240208 eth0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2498 > y.y.y.y.0: udp 0 (ttl 51, id
44437) (ttl 51, id 17453)
                    4500 0044 442d 0000 3301 feaf yyyy yyyy
                    xxxx xxxx 0303 739c 0000 0000 4500 001c
```

```
ad95 0000 3311 955f xxxx xxxx yyyy yyyy
09c2 0000 0008 b13f dd2c 2a16 38e1 7646
7aaa 9d41
```

From the tcpdump trace we can see that the offending packet's IP header and the first 8 data bytes were echoed correctly. Right after those, 12 bytes were padded, that came from nowhere.

The next example is with Foundry Network's BigIron 8000 running software version 6.6.05T51. With this test I have sent a UDP datagram with 80 bytes of data to a closed UDP port on the BigIron 8000:

```
[root@godfather /root]# hping2 -2 -c 3 -d 80 y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y (ppp0 y.y.y.y ): udp mode set, 28 headers + 80 data
bytes
ICMP Port Unreachable from y.y.y.y (y.y.y.y)
ICMP Port Unreachable from y.y.y.y (y.y.y.y)
ICMP Port Unreachable from y.y.y.y (y.y.y.y)

--- y.y.y.y hping statistic ---
3 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather /root]#


11:40:36.694235 ppp0 > x.x.x.x.2779 > y.y.y.y.0: udp 80 (ttl 64, id
25211)
                  4500 006c 627b 0000 4011 2e7a xxxx xxxx
                  yyyy yyyy 0adb 0000 0058 3d09 5858 5858
                  5858 5858 5858 5858 5858 5858 5858 5858
                  5858 5858 5858 5858 5858 5858 5858 5858
                  5858 5858 5858 5858 5858 5858 5858 5858
                  5858 5858 5858 5858 5858 5858 5858 5858
                  5858 5858 5858 5858 5858 5858

11:40:37.913018 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2779 > y.y.y.y.0: udp 80 (ttl 52, id
25211) (ttl 52, id 60504)
                  4500 0044 ec58 0000 3401 b0d4 yyyy yyyy
                  xxxx xxxx 0303 edf3 0000 0000 4500 006c
                  627b 0000 3411 3a7a xxxx xxxx yyyy yyyy
                  0adb 0000 0058 3d09 1c1d 1e1f 2021 2223
                  2425 2627
```

Again, the offending packet's IP Header and the first 8 data bytes are quoted correctly. 12 data bytes are padded right after.

A nice pattern that allows us to identify Foundry Network's networking devices.

6.17 ICMP Error Message Echoing Integrity (Tested with ICMP Port Unreachable)

When sending back an ICMP error message, some stack implementations may alter the original IP header, which is echoed back with the ICMP error message.

If an attacker examines the types of alternation that have been made to the headers, he may be able to make certain assumptions about the target operating system.

The only two field values we expect to be changed are the IP time-to-live field value and the IP header checksum. The TTL field value changes because the field is decremented by one, each time the IP Header is processed. The IP header checksum is recalculated each time the IP TTL field value decremented.

Fyodor gives the following examples in his article "Remote OS detection via TCP/IP Stack Finger Printing"[44]:

> "For example, AIX and BSDI send back an IP 'total length' field that is 20 bytes too high. Some BSDI, FreeBSD, OpenBSD, ULTRIX, and VAXen change the IP ID that you sent them. While the checksum is going to change due to the changed TTL anyway, there are some machines (AIX, FreeBSD, etc.) which send back an inconsistent or 0 checksum. Same thing goes with the UDP checksum."

This section deals with the ICMP Port Unreachable error message.

AIX 4.2.1, 4.3, 4.3 tlx pack 2
In the next example I have sent a UDP datagram to a closed UDP port on an AIX 4.3 machine using HPING2. This is the tcpdump trace:

```
12:33:17.319275 ppp0 > x.x.x.x.2160 > y.y.y.y.0: udp 0 [tos 0x10]  (ttl
64, id 47349)
                    4510 001c b8f5 0000 4011 9bea xxxx xxxx
                    yyyy yyyy 0870 0000 0008 d18c

12:33:17.614823 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2160 > y.y.y.y.0: udp 0 [tos 0x10]
(ttl 49, id 47349, bad cksum aaea!) [tos 0x10]  (ttl 241, id 17965)
                    4510 0038 462d 0000 f101 5da6 yyyy yyyy
                    xxxx xxxx 0303 f470 0000 0000 4510 0030
                    b8f5 0000 3111 aaea xxxx xxxx yyyy yyyy
                    0870 0000 0008 0000
```

We can identify several changed between the original IP Headers to the echoed ICMP Header with the ICMP port unreachable error message.

- **IP Total Length Field** - The total length field with the original UDP datagram equal to 28 bytes. With the echoed original IP header this value was changed to 48 bytes. 20 bytes more than the original UDP datagram's length.
- **IP TTL Field value** - With the ICMP error message this value is set to the value, which reached its final destination (with this example the targeted host). When it reached it target the TTL was set to 49. We also learn the target is 64-49 = 15 hops away.
- **IP Header Checksum** - The IP Header checksum was changed because the IP Total Length filed value and the IP TTL field value were changed.

[44] http://www.insecure.org/nmap/nmap-fingerprinting-article.html

- **UDP Header Checksum** – The UDP header checksum with the echoed information equal to zero.

AIX 4.1
In the next example I have sent a UDP datagram to a closed UDP port on an AIX 4.1 machine using HPING2. This is the tcpdump trace:

```
00:56:07.894612 ppp0 > x.x.x.x.1594 > y.y.y.y.0: udp 0 [tos 0x8]  (ttl
64, id 2153)
                    4508 001c 0869 0000 4011 c54f xxxx xxxx
                    yyyy yyyy 063a 0000 0008 4c93

00:56:08.204551 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.1594 > y.y.y.y.0: udp 0 [tos 0x8]
(ttl 47, id 2153, bad cksum d64f!) [tos 0x8]  (ttl 239, id 1065)
                    4508 0038 0429 0000 ef01 1a83 yyyy yyyy
                    xxxx xxxx 0303 aa13 0000 0000 4508 0030
                    0869 0000 2f11 d64f xxxx xxxx yyyy yyyy
                    063a 0000 0008 4c93
```

We can identify several changed between the original IP Headers to the echoed ICMP Header with the ICMP port unreachable error message.

- **IP Total Length Field** - The total length field with the original UDP datagram equal to 28 bytes. With the echoed original IP header this value was changed to 48 bytes, 20 bytes more than the original UDP datagram's length.
- **IP TTL Field value** - With the ICMP error message this value is set to the value, which reached its final destination (with this example the targeted host). When it reached it target the TTL was set to 47. We also learn the target is 64-47 = 17 hops away.
- **IP Header Checksum** - The IP Header checksum was changed because the IP Total Length filed value and the IP TTL field value were changed.

ICMP Error Message Echoing Integrity with different 4.x versions of AIX
In contrast to AIX version 4.3 and 4.2.1 AIX version 4.1 use the original UDP Checksum. This detail helps us to differentiate between the different versions of AIX.

BSDI 4.x
In the next example I have sent, again, a UDP datagram to a close UDP port, this time on a BSDI 4.1 machine. The following is the tcpdump trace:

```
01:01:11.128420 ppp0 > x.x.x.x.2933 > y.y.y.y.0: udp 0 [tos 0x8]  (ttl
64, id 49317)
                    4508 001c c0a5 0000 4011 9209 xxxx xxxx
                    yyyy yyyy 0b75 0000 0008 cc4e

01:01:11.484552 ppp0 < y.y.y.y.4 > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2933 > y.y.y.y.0: udp 0 [tos 0x8]
(ttl 53, id 49317, bad cksum 0!) (ttl 242, id 16127)
                    4500 0038 3eff 0000 f201 61ab yyyy yyyy
```

```
xxxx xxxx 0303 c226 0000 0000 4508 0030
c0a5 0000 3511 0000 xxxx xxxx yyyy yyyy
0b75 0000 0008 cc4e
```

Again several changed were made to the original IP Header:

- **IP Total length** - With the echoed IP Header this field value was changed from the original 28 bytes to 48 bytes. 20 bytes more than the original.
- **IP TTL Field Value** – Changed according to the hop count. Was equal to 53 when arrived to its destination. The target is 64 – 53 = 11 hops away.
- **IP Header Checksum** – changed now it equal to zero!

FreeBSD 3.x up to 4.1.1 (not including)
The next example is with FreeBSD 4.1:

```
00:52:19.055758 ppp0 > x.x.x.x.1393 > y.y.y.y.0: udp 0 [tos 0x8]  (ttl
64, id 58965)
                4508 001c e655 0000 4011 3f63 xxxx xxxx
                yyyy yyyy 0571 0000 0008 a55e

00:52:19.464548 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.1393 > y.y.y.y.0: udp 0 [tos 0x8]
(ttl 47, id 21990, bad cksum 5063!) (ttl 238, id 27639)
                4500 0038 6bf7 0000 ee01 0bbd yyyy yyyy
                xxxx xxxx 0303 87f3 0000 0000 4508 001c
                55e6 0000 2f11 5063 xxxx xxxx yyyy yyyy
                0571 0000 0008 0000
```

- The **IP Identification** field value is changed. This field is constructed with 16bit. The first 8 bits changed places with the second pair of 8 bits constructing this field. With the original datagram this field value was e655, with the echoed IP header it is 55e6[45].
- **The IP TTL field value** has changed. The target is 64 – 47 = 17 hops away.
- **The IP Header Checksum** have changed because of the parameters were changed as well.
- **The UDP checksum** is changed and now it equal to zero!

Operating System	DF Bit set with the Reply?	IP Total Length	IP Identification	IP TTL field value	IP Header Checksum	UDP Checksum
LINUX Kernel 2.2.x	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
LINUX Kernel 2.4	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same

[45] http://www.freebsd.org/cgi/query-pr.cgi?pr=16240 ; Patches were issued. This is fixed with FreeBSD 4.1.1.

ICMP Usage in Scanning
Version 2.5

FreeBSD 4.0	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
FreeBSD 4.11	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
BSDI 4.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed. Now equals to ZERO!	Same
Sun Solaris 2.6	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Sun Solaris 2.7	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Sun Solaris 2.8[46]	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
HPUX 11.0	No -> Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Compaq Tru64	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
DG-UX 5.6	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
AIX 4.3 fp2, 4.3, 4.2.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed because of new parameters.	Changed. Now equal to ZERO!
AIX 4.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed because of new parameters.	Same

[46] The DF Bit is set.

Operating System	DF Bit set with the Reply?	IP Total Length	IP Identification	IP TTL field value	IP Header Checksum	UDP Checksum
ULTRIX	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count	Changed. Now equals to ZERO!	Changed. Now equal to ZERO!
OpenVMS	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count	Changed. Now equals to ZERO!	Changed. Now equal to ZERO!
Microsoft windows 98						
Mirosoft Windows 98SE	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows ME	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows NT 4	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows 2000 Family	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same

Table 14: ICMP Error Message Echoing Integrity

6.18 Novell Netware Echoing Integrity Bug with ICMP Fragment Reassembly Time Exceeded

Novell Netware operating systems have a unique pattern with ICMP Fragment Reassembly Time Exceeded error messages they produce.

In general, when an ICMP error message is produced, the offending packet's IP Header + at least 8 bytes of data are echoed with the error message.

If we examine closely the next example, we can see that the Offending packet's IP TTL field value echoed back is zero.

We expect this value to decrement from the value initially assigned, but not to be zero. Since this value should change from one hop to another, the Checksum need to be recalculated each time. With the Novell Netware error message we can see that the Checksum echoed is miscalculated.

...And again this is a Fragment Reassembly Time Exceeded ICMP error message and not an ICMP Time Exceeded in Transit error message.

The next example is with Novell Netware 5.1:

```
[root@godfather bin]# hping2 -c 1 -x -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y (ppp0 y.y.y.y): NO FLAGS are set, 40 headers + 0 data
bytes

--- y.y.y.y hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather bin]#
```

The Trace:

```
20:12:28.008893 ppp0 > x.x.x.x.1865 > y.y.y.y.0: .
687160929:687160929(0) win 512 (frag 58586:20@0+) (DF) (ttl 64)
                        4500 0028 e4da 6000 4006 c236 xxxx xxxx
                        yyyy yyyy 0749 0000 28f5 3e61 669e 9f15
                        5000 0200 c5d2 0000

20:12:41.313202 ppp0 < y.y.y.y > x.x.x.x: icmp: ip reassembly time
exceeded Offending pkt: [|tcp] (frag 58586:20@0+) (DF) [ttl 0] (bad
cksum d336!) (ttl 111, id 9591)
                        4500 0038 2577 0000 6f01 b28f yyyy yyyy
                        xxxx xxxx 0b01 b55f 0000 0000 4500 0028
                        e4da 6000 0006 d336 xxxx xxxx yyyy yyyy
                        0749 0000 28f5 3e61
```

This unique pattern enables us to determine if the operating system in question is a Novell Netware or other with one datagram only.

6.19 The Precedence bits with ICMP Error Messages (Identifying LINUX)

Each IP Datagram has an 8-bit field called the "TOS Byte", which represents the IP support for prioritization and Type-of-Service handling.

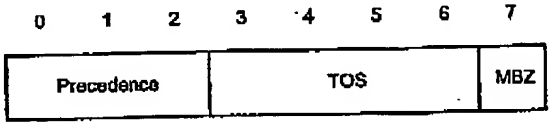

Figure 13: The Type of Service Byte

The "TOS Byte" consists of three fields.

The "Precedence field", which is 3-bit long, is intended to prioritize the IP Datagram. It has eight levels of prioritization[47]:

Precedence	Definition
0	Routine (Normal)
1	Priority
2	Immediate
3	Flash
4	Flash Override
5	Critical
6	Internetwork Control
7	Network control

Table 15: Precedence Field Values

Higher priority traffic should be sent before lower priority traffic.

The second field, 4 bits long, is the "Type-of-Service" field. It is intended to describe how the network should make tradeoffs between throughput, delay, reliability, and cost in routing an IP Datagram.

The last field, the "MBZ" (most be zero), is unused and most be zero. Routers and hosts ignore this last field. This field is 1 bit long.

RFC 1122 Requirements for Internet Hosts -- Communication Layers, states:
"The Precedence field is intended for Department of Defense applications of the Internet protocols. The use of non-zero values in this field is outside the scope of this document and the IP standard specification. Vendors should consult the Defense Communication Agency (DCA) for guidance on the IP Precedence field and its implications for other protocol layers. However, vendors should note that the use of precedence will most likely require that its value be passed between protocol layers in just the same way as the TOS field is passed".

Other precedence information is available with RFC 1812 Requirements for IP Version 4 Routers:
"4.3.2.5 TOS and Precedence
...

ICMP Source Quench error messages, if sent at all, MUST have their IP Precedence field set to **the same value as the IP Precedence field in the packet that provoked the sending of the ICMP Source Quench message. All other ICMP error messages** (Destination Unreachable, Redirect, Time Exceeded, and Parameter Problem) **SHOULD** have their precedence value set to **6** (INTERNETWORK CONTROL) or **7** (NETWORK CONTROL). The IP Precedence value for these error messages MAY be settable".

With the operating systems I have checked, nearly all of them used the value of 0x00 for the Precedence field (bits).

All but LINUX

[47] RFC 791 – Internet Protocol, http://www.ietf.org/rfc/rfc791.txt.

Fyodor had outlined in his paper "Remote OS Identification by TCP/IP Fingerprinting"[48] the fact that LINUX is using the value of 0xc0 (an unused precedence value) as its TOS byte value with ICMP Port Unreachable error messages.

In the next example we have sent one UDP packet destined to port 50 (which is closed on the destination machine) from one LINUX machine to another, both running Redhat LINUX 6.1:

```
[root@stan /root]# hping2 -2 192.168.5.5 -p 50 -c 1
default routing not present
HPING 192.168.5.5 (eth0 192.168.5.5): udp mode set, 28 headers + 0 data
bytes
ICMP Port Unreachable from 192.168.5.5  (kenny.sys-security.com)

--- 192.168.5.5 hping statistic ---
1 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms


Kernel filter, protocol ALL, raw packet socket
Decoding Ethernet on interface eth0
03/12-12:54:47.274096 192.168.5.1:2420 -> 192.168.5.5:50
UDP TTL:64 TOS:0x0 ID:57254
Len: 8

03/12-12:54:47.274360 192.168.5.5 -> 192.168.5.1
ICMP TTL:255 TOS:0xC0 ID:0
DESTINATION UNREACHABLE: PORT UNREACHABLE
00 00 00 00 45 00 00 1C DF A6 00 00 40 11 0F D4  ....E.......@...
C0 A8 05 01 C0 A8 05 05 09 74 00 32 00 08 6A E1  .........t.2..j.
```

This abnormality with LINUX is not only limited to ICMP Destination Unreachable Port Unreachable error messages.

Lets examine the next trace:

```
00:30:08.339498 < x.x.x.x > y.y.y.y: ip-proto-72 0 (ttl 49, id 38624)
                  4500 0014 96e0 0000 3148 f4bf xxxx xxxx
                  yyyy yyyy

00:30:08.339559 > y.y.y.y > x.x.x.x: icmp: y.y.y.y protocol 72
unreachable Offending pkt: x.x.x.x > y.y.y.y: ip-proto-72 0 (ttl 49, id
38624) [tos 0xc0]  (ttl 255, id 37)
                  45c0 0044 0025 0000 ff01 bcd1 yyyy yyyy
                  xxxx xxxx 0302 fb1a 0000 0000 4500 0014
                  96e0 0000 3148 f4bf xxxx xxxx yyyy yyyy
                  0050 d909 621b 96f7 0000 0000 5004 0000
                  df71 0000
```

The ICMP error message produced by a LINUX machine based on Kernel 2.2.14, is Destination Unreachable Protocol Unreachable (Type 3 Code 2). As it can be seen the TOS Byte value that was used is again 0xc0. Which is an unused Precedence bits value.

[48] This fact was discovered by Fyodor. http://www.insecure.org/nmap/nmap-fingerprinting-article.html

LINUX embraced the behavior RFC 1812 suggested and sends all his ICMP error messages with the Precedence field value sent to 0xc0 (value of 6).

Just to remind the reader – LINUX is not a router.

6.20 TOS Bits (=field) Echoing with ICMP Error

RFC 1394 specify that an ICMP error message be always sent with the default TOS field value of 0000 (TOS field=TOS bits in the TOS Byte).

When an offending packet with a TOS field value of 0000 is eliciting an ICMP error message from an offended host, the TOS field value with all the operating systems I have checked will be set to 0000.

If we will pay attention to the TOS Byte we will see that LINUX and several routers will use the value of 0xc0 for the precedence field (see section 6.14 The Precedence bits with ICMP Error Messages for the explanation).

What will happen if the TOS field with the offending packet will be set to a value different than the default (0000)?

We will have several operating systems that will echo the TOS field back with the ICMP error message.

Our first example is with an AIX 4.3 machine. where a UDP datagram is sent with a TOS field value of 0x10 hex:

```
12:33:17.319275 ppp0 > x.x.x.x.2160 > y.y.y.y.0: udp 0 [tos 0x10]  (ttl
64, id 47349)
                    4510 001c b8f5 0000 4011 9bea xxxx xxxx
                    yyyy yyyy 0870 0000 0008 d18c

12:33:17.614823 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 0
unreachable Offending pkt: x.x.x.x.2160 > y.y.y.y.0: udp 0 [tos 0x10]
(ttl 49, id 47349, bad cksum aaea!) [tos 0x10]  (ttl 241, id 17965)
                    4510 0038 462d 0000 f101 5da6 yyyy yyyy
                    xxxx xxxx 0303 f470 0000 0000 4510 0030
                    b8f5 0000 3111 aaea xxxx xxxx yyyy yyyy
                    0870 0000 0008 0000
```

As it can be seen from the trace, the TOS field value was echoed back by the AIX machine. This was tested against AIX 4.1, 4.2.1, 4.3, 4.3 fix pack2.

The next example is with DGUX 5.6:

```
12:58:57.663517 ppp0 > x.x.x.x.1074 > y.y.y.y.11: udp 0 [tos 0x8]  (ttl
64, id 47314)
                    4508 001c b8d2 0000 4011 a037 xxxx xxxx
                    yyyy yyyy 0432 000b 0008 d9e1

12:58:57.984820 ppp0 < 134.210.1.200 > x.x.x.x.: icmp: y.y.y.y.200 udp
port 11 unreachable Offending pkt: x.x.x.x.1074 > y.y.y.y.11: udp 0
[tos 0x8]  (ttl 52, id 47314) [tos 0x8]  (ttl 52, id 16984)
```

```
4508 0038 4258 0000 3401 22a6 yyyy yyyy
d508 c41c 0303 f8b7 0000 0000 4508 001c
b8d2 0000 3411 ac37 xxxx xxxx yyyy yyyy
0432 000b 0008 0000
```

How can we differentiate between DGUX and AIX? If we will pay attention to the echoing integrity. AIX 4.x sets the IP total length field value, with the echoed offending IP Header, to a value 20 bytes higher than the original. DGUX quote this field value correctly.

The last operating system, which I have found echoing the TOS field value with its ICMP error messages, is LINUX operating systems based on Kernel 2.2.x & 2.4 (the versions of the Kernel that I have tested):

```
00:50:43.759906 ppp0 > x.x.x.x.1952 > y.y.y.y.0: udp 0 [tos 0x10]  (ttl
64, id 15952)
                    4510 001c 3e50 0000 4011 e6b2 xxxx xxxx
                    yyyy yyyy 07a0 0000 0008 a27f

00:50:44.154556 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y.211 udp port 0
unreachable Offending pkt: x.x.x.x.1952 > y.y.y.y.0: udp 0 [tos 0x10]
(ttl 47, id 15952) [tos 0xd0]  (ttl 238, id 54662)
                    45d0 0038 d586 0000 ee01 a0af yyyy yyyy
                    xxxx xxxx 0303 52d5 0000 0000 4510 001c
                    3e50 0000 2f11 f7b2 xxxx xxxx yyyy yyyy
                    07a0 0000 0008 a27f
```

Another unique pattern with LINUX is setting the Precedence field value to 0xc0 with ICMP error messages. This helps us to differentiate LINUX from the other operating systems that echo the TOS field value.

While LINUX embraced RFC 1812 instructions for routers regarding the TOS and Precedence fields, the other operating systems that echo the TOS field value don't seem to have a good excuse for doing so.

6.21 DF Bit Echoing with ICMP Error Messages

We already have the DF Bit Echoing method with ICMP Query message types (& Replies); I was thinking why this couldn't happen with ICMP Error Messages as well?

What will happen if we will set the DF bit with an offending packet that will generate an ICMP Error message? Will the DF Bit be set with the ICMP Error Message?

In the next example, a UDP datagram is sent to a closed UDP port, to elicit an ICMP Port Unreachable error message. The DF bit is set with the offending datagram. As it can be seen the DF bit is set with the ICMP error message the FreeBSD 4.1.1 machine, which was the target system issued back.

```
[root@godfather /root]# hping2 -2 -p 2000 -c 2 -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y (ppp0 y.y.y.y): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from y.y.y.y  (host_address)
ICMP Port Unreachable from y.y.y.y  (host_address)

--- y.y.y.y hping statistic ---
```

```
2 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather /root]#

00:31:29.805075 ppp0 > x.x.x.x.1403 > y.y.y.y.2000: udp 0 (DF) (ttl 64,
id 19417)
                  4500 001c 4bd9 4000 4011 452b xxxx xxxx
                  yyyy yyyy 057b 07d0 0008 48c6

00:31:30.103692 ppp0 < 18.170.1.79 > x.x.x.x: icmp: y.y.y.y udp port
2000 unreachable Offending pkt: x.x.x.x.1403 > y.y.y.y.2000: udp 0 (DF)
(ttl 45, id 19417) (DF) (ttl 238, id 47017)
                  4500 0038 b7a9 4000 ee01 2b4e yyyy yyyy
                  xxxx xxxx 0303 efa9 0000 0000 4500 001c
                  4bd9 4000 2d11 582b xxxx xxxx yyyy yyyy
                  057b 07d0 0008 0000
```

We can distinguish between the group of operating systems, which will echo back the DF bit with their replies, to the group of operating systems that will not.

The next example is with Microsoft Windows ME:

```
00:49:45.853751 ppp0 > x.x.x.x.1580 > y.y.y.y.10: udp 0 (DF) (ttl 64,
id 63227)
                  4500 001c f6fb 4000 4011 730a xxxx xxxx
                  yyyy yyyy 062c 000a 0008 28dd

00:49:46.173681 ppp0 < 212.150.102.96 > x.x.x.x: icmp: y.y.y.y udp port
10 unreachable Offending pkt: x.x.x.x.1580 > y.y.y.y.10: udp 0 (DF)
(ttl 55, id 63227) (ttl 119, id 430)
                  4500 0038 01ae 0000 7701 714c yyyy yyyy
                  xxxx xxxx 0303 cde1 0000 0000 4500 001c
                  f6fb 4000 3711 7c0a xxxx xxxx yyyy yyyy
                  062c 000a 0008 28dd
```

Among the operating systems I have checked LINUX machines based on Kernel 2.2.x / 2.4.x, ULTRIX, Novell Netware, and Microsoft Windows 98/98SE/ME/NT4SP6A/Windows 2000 Family, will not echo back the DF bit with their ICMP Error messages.

How can we distinguish between the operating systems in the non-DF echoing group? Since Linux is using the value of 0xc0 hex for his Precedence Bits field value for all ICMP Error messages we can separate it instantly.

```
00:25:17.203727 ppp0 > x.x.x.x.1421 > y.y.y.y.2000: udp 0 (DF) (ttl 64,
id 11969)
                  4500 001c 2ec1 4000 4011 b938 xxxx xxxx
                  yyyy yyyy 058d 07d0 0008 9fa9

00:25:17.573698 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port 2000
unreachable Offending pkt: x.x.x.x.1421 > y.y.y.y.2000: udp 0 (DF) (ttl
45, id 11969) [tos 0xc0]  (ttl 236, id 38250)
                  45c0 0038 956a 0000 ec01 e5c2 yyyy yyyy
                  xxxx xxxx 0303 4fee 0000 0000 4500 001c
                  2ec1 4000 2d11 cc38 xxxx xxxx yyyy yyyy
                  058d 07d0 0008 9fa9
```

ULTRIX echo integrity is not that good. The offending packet echoing will set both the IP Header Checksum and the Original UDP Checksum to zero. It will also miscalculate the IP ID field value and will flip the first 8 bits with the second one, creating a false value for it:

```
00:29:05.013726 ppp0 > x.x.x.x.1188 > y.y.y.y.2000: udp 0 (DF) (ttl 64,
id 34921)
                    4500 001c 8869 4000 4011 5f85 xxxx xxxx
                    yyyy yyyy 04a4 07d0 0008 a087

00:29:05.383686 ppp0 < 194.47.250.222 > x.x.x.x: icmp: y.y.y.y udp port
2000 unreachable Offending pkt: x.x.x.x.1188 > y.y.y.y.2000: udp 0 (ttl
45, id 27016, bad cksum 01) (ttl 236, id 9736)
                    4500 0038 2608 0000 ec01 55da yyyy yyyy
                    xxxx xxxx 0303 c1e7 0000 0000 4500 001c
                    6988 0000 2d11 0000 xxxx xxxx yyyy yyyy
                    04a4 07d0 0008 0000
```

This will leave us with Novell Netware and the various Microsoft Windows Operating Systems.

As discussed in section 6.17 "*Novell Netware Echoing Integrity Bug with ICMP Fragment Reassembly Time Exceeded*", when a Novell Netware operating system issue an ICMP Time Exceeded error message it will zero out on the echoed offending packet the IP TTL field value. We will use this information and send an offending packet to the questioned operating systems that will elicit an ICMP Time Exceeded error message from the quesitoned OSs.

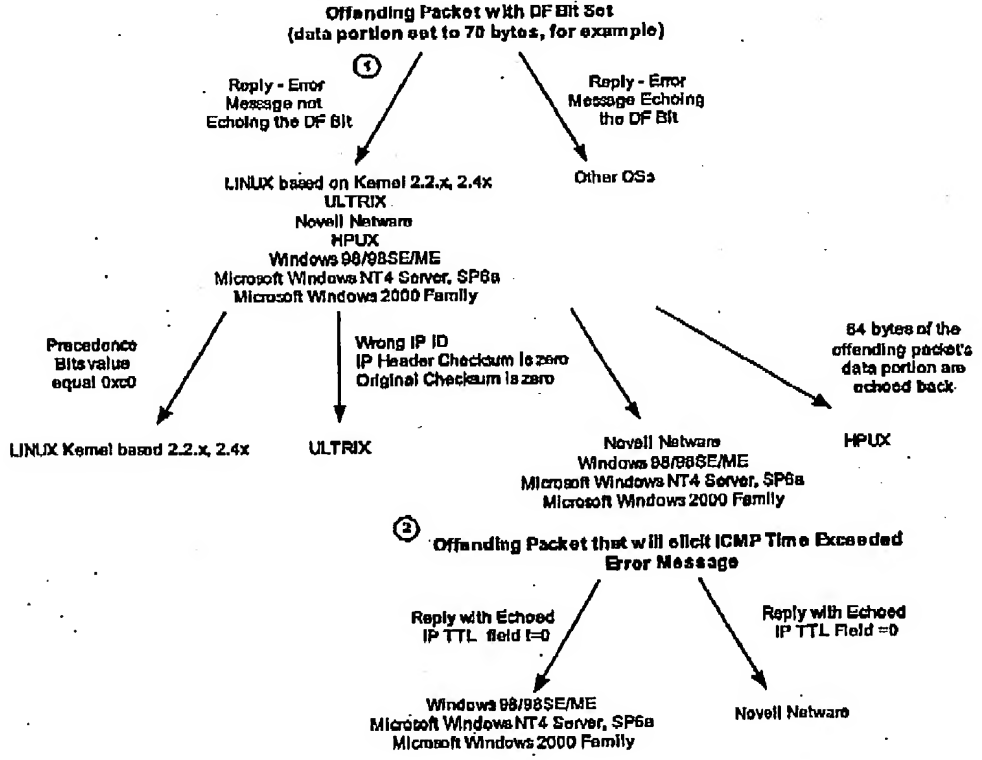

Diagram 12: DF Bit Echoing with ICMP Error Messages

We can take a second approach using the ICMP Fragment Reassembly Time Exceeded error message. We will send an offending packet with the DF bit set that will elicit an ICMP Fragment Reassembly Time Exceeded error message back from the targeted machine. Novell Netware, LINUX based Kernel 2.2.x and 2.4x-t, and the various Microsoft Windows operating systems will set the DF bit with their replies. LINUX and Novell have their unique fingerprinting with ICMP Fragment Reassembly Time Exceeded error messages, enabling us to isolate the Microsoft based operating systems machines.

HP-UX 11.x based machines will have a unique behavior when the PMTU discovery process based on ICMP Echo Requests is enabled (by default). In the next example I have sent a UDP datagram to port 53 (DNS) of the targeted HPUX machine.

```
[root@godfather /root]# hping2 -2 -p 53 -c 2 -y y.y.y.y
ppp0 default routing interface selected (according to /proc)
HPING y.y.y.y (ppp0 y.y.y.y): udp mode set, 28 headers + 0 data bytes
ICMP Port Unreachable from y.y.y.y (unknown host name)
ICMP Port Unreachable from y.y.y.y (unknown host name)

--- y.y.y.y hping statistic ---
2 packets tramitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
[root@godfather /root]#
```

```
00:45:02.490445 ppp0 > x.x.x.x.codasrv > y.y.y.y.domain: 0 [0q] (0)
(DF) (ttl 64, id 7454)
                         4500 001c 1d1e 4000 4011 e708 xxxx xxxx
                         yyyy yyyy 0980 0035 0008 bf7e
```

As an instant reply the PMTU discovery process which is based upon ICMP Echo request(s) is started:

```
00:45:03.113686 ppp0 < y.y.y.y > x.x.x.x: icmp: echo request (DF) (ttl
242, id 25153)
                         4500 05dc 6241 4000 f201 ea34 yyyy yyyy
                         xxxx xxxx 0800 7e52 9abc def0 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
                         0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000
```

My LINUX machine replied with ICMP Echo Reply:

```
00:45:03.113787 ppp0 > x.x.x.x > y.y.y.y: icmp: echo reply (ttl 255, id
98)
                 4500 05dc 0062 0000 ff01 7f14 xxxx xxxx
                 yyyy yyyy 0000 8652 9abc def0 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
                 0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
                    0000 0000 0000 0000 0000 0000 0000 0000
                    0000 0000 0000 0000 0000 0000
```

The first ICMP Port Unreachable error message arrives without the DF bit set:

```
00:45:03.123692 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port domain
unreachable Offending pkt: x.x.x.x.codasrv > y.y.y.y.domain: 0 [0q] (0)
(DF) (ttl 51, id 7454) (ttl 242, id 25154)
                    4500 0038 6242 0000 f201 2fd8 yyyy yyyy
                    xxxx xxxx 0303 33c1 0000 0000 4500 001c
                    1d1e 4000 3311 f408 xxxx xxxx yyyy yyyy
                    0980 0035 0008 bf7e
```

A second UDP datagram is sent:

```
00:45:03.493752 ppp0 > x.x.x.x.codasrv-se > y.y.y.y.domain: 56810+ (0)
(DF) (ttl 64, id 59904)
                    4500 001c ea00 4000 4011 1a26 xxxx xxxx
                    yyyy yyyy 0981 0035 0008 bf7d
```

The ICMP Port Unreachable error message that was sent for the second UDP datagram now sets the DF bit as part of the PMTU discovery process maintenance:

```
00:45:03.813687 ppp0 < y.y.y.y > x.x.x.x: icmp: y.y.y.y udp port domain
unreachable Offending pkt: x.x.x.x.codasrv-se > y.y.y.y.domain: 26990
op5+ [b2&3=0x2d61] [29188a] [25700q] [24946n] [28769au] (0) (DF) (ttl
51, id 59904) (DF) (ttl 242, id 25155)
                    4500 0038 6243 4000 f201 efd6 yyyy yyyy
                    xxxx xxxx 0303 33c1 0000 0000 4500 001c
                    ea00 4000 3311 2726 xxxx xxxx yyyy yyyy
                    0981 0035 0008 bf7d
```

This also means that with the regular behavior with HPUX 11.x it will not echo back the DF bit. Also, if you are sending only one offending datagram to the targeted HPUX 11.x based machine, you will not be able to see the change.

So how can we distinguish HPUX from the other operating systems?

HPUX based operating system(s) machines will echo up to 64 bytes of the offending packet's data portion. By sending a bigger offending datagram (for example with 80 bytes of data portion) we can examine which of the operating systems in question, which do not set the DF bit with the ICMP error message, will echo 64 bytes of the data portion (or even more than 8 and will not set the the precedence bits to 0xc0).

Not that useful fingerprinting method(s)

6.22 Unusual Big ICMP Echo Request

What would happen if we would send unusual big ICMP Echo message that would require its fragmentation? Would the queried operating systems will process the query correctly and produce an accurate reply?

```
[root@aik /root]# ping -s 1500 x.x.x.x
```

```
PING x.x.x.x (x.x.x.x) from y.y.y.y : 1500(1528) bytes of data.
1508 bytes from x.x.x.x: icmp_seq=0 ttl=241 time=1034.7 ms
1508 bytes from host_address (x.x.x.x): icmp_seq=2 ttl=241 time=1020.0
ms
1508 bytes from host_address (x.x.x.x): icmp_seq=3 ttl=241 time=1090.4
ms
1508 bytes from host_address (x.x.x.x): icmp_seq=5 ttl=241 time=1060.0
ms

--- x.x.x.x ping statistics ---
8 packets transmitted, 5 packets received, 37% packet loss
round-trip min/avg/max = 1000.2/1041.0/1090.4 ms
[root@aik /root]#
```

As it seems all the probed operating systems I have tested behaved correctly processing the query and sending the reply back.

What else can assist us with this kind of query?
The DF (Don't Fragment) bit.

Some operating systems would process the query and set the don't fragment bit on the fragments of the reply like we have outlined in the "DF Bit Playground" section. These operating systems would be Sun Solaris, and HP-UX 10.30 & 11.0x[49].

We can use other methods, which does not generate the kind of noise this method generates. Basically there is no reason for this size of ICMP Echo request. This should trigger IDS systems immediately that something suspicious is happening.

[49] Please refer to section 6.2 for more information.

7.0 Filtering ICMP on your Filtering Device to Prevent Scanning Using ICMP

7.1 Inbound

An example of incoming ICMP traffic that should be blocked in order to *prevent scanning techniques that were outlined in this paper* might be:

- ICMP Echo (used for Host Detection, traceroute, Inverse Mapping, and Operating System Fingerprinting)
- ICMP Echo Reply (used for Inverse Mapping)
- ICMP Time Stamp Requests (used for Host Detection, Operating System Fingerprinting)
- ICMP Address Mask Request (used for Host Detection, Operating System Fingerprinting)
- All ICMP Message Types (Inverse Mapping Technique)
- ICMP Error messages (Operating System Fingerprinting)
- All ICMP Message Types should be blocked in order to prevent the fingerprinting techniques I have outlined in this research paper.

- You should also block the IP directed broadcast on your border router.
- Deny access to your Broadcast and Network addresses from the Internet.

If you look closely at this list, it is all ICMP Message types, whether query types or error types.

7.2 Outbound

There are people who claim that any traffic type of ICMP should be allowed from a protected network to the Internet. This is not true. Filtering the incoming traffic does not mean we are protected from some of the security hazards I outlined in this paper.

7.2.1 ICMP ECHO Reply (Type 0)
Used to map a host using Host Detection.

7.2.2 ICMP Destination Unreachable Messages
I have demonstrated that host detection can be done with bad IP Header packets, which elicit various ICMP Parameter Problem and ICMP Destination Unreachable error messages from the probed machines and draw the attacked network topology.

7.2.3 ICMP "Fragmentation Needed and Don't Fragment Bit was Set"
See section 3.5

7.2.4 ICMP ECHO (Type 8)
We have to have a Stateful filtering device that would perform Stateful Inspection with ICMP in order to let ICMP ECHO Requests out, and receive only the corresponding ICMP ECHO Replies.

The current state with filtering devices is not that bright. Most of them do not perform Stateful Inspection with the ICMP protocol. Allowing ICMP ECHO Replies inside our protected network is very dangerous and is not worth it.

Unless you use a Stateful filtering device with the ICMP protocol don't let ICMP ECHO Replies into your protected network. This would make your requests useless so you better block them.

7.2.5 ICMP Time to Live Exceeded In Transit (Type 11 Code 0)
To eliminate traceroute and Reverse Mapping techniques we do not want to let a Time-to-Live Exceeded code 0 messages go back to the malicious computer attacker.

7.2.6 ICMP Fragmentation Reassembly Time Exceeded (Type 11 Code 1)
By blocking this ICMP type we eliminate the usage of a Host Detection technique, which sends only few fragments, form a fragmented datagram, and force the probed host to send us an ICMP Fragmentation Reassembly Time Exceeded error message back revealing his existence.

7.2.7 ICMP Parameter Problem
We have demonstrated that host detection can be made with bad IP Header packets, which would elicit various ICMP Parameter Problem and ICMP Destination Unreachable error messages from the probed machines.

7.2.8 ICMP Time Stamp Request & Reply
Time Stamp requests & replies can be used for Host Detection and Inverse Mapping.

7.2.9 ICMP Address Mask Request and Reply
Address Mask request & reply can be used for host detection and Inverse Mapping.

7.2.10 The liability Question
System administrator / Network administrator don't want to be held liable for an attack generated from there network by an abusive user (or a malicious computer attacker using a compromised system within the network). Therefore blocking some types of ICMP traffic from the protected network to the outside world is recommended for liability reasons:

- Destination Unreachable Codes 2-4
  - ICMP Destination Unreachable error messages 2-4 ("Port Unreachable", "Protocol Unreachable" and "Fragmentation Needed and DF Flag was Set") Is a group of messages that are hard error conditions and when received should terminate a connection.

    This allow an attacker to send *fake* Destination Unreachable codes 2-4 to terminate valid connections between the attacked target and other hosts on the void.

    Old TCP/IP implementations terminat TCP connections when receiving those error messages. Modern TCP/IP implementations no longer terminate a TCP connection when receiving those error messages
- Source Quench messages
  - Since hosts still react to Source Quenches by slowing communication, they can be used as a Denial-of-Service measure.
- Redirect messages
  - If you can forge ICMP Redirect packets, and if your target host pays attention to them - ICMP Redirects may be employed for denial of service attacks, where a

host is sent a route that loses it connectivity, or is sent an ICMP Network Unreachable packet telling it that it can no longer access a particular network.

This means that all outbound ICMP traffic should be disallowed.

7.3 Other Considerations

If you want to maintain strong ICMP filtering rules with your Firewall/Filtering-Device I suggest you block all incoming ICMP traffic except for Type 3 Code 4, which is used by the Path MTU Discovery process[50]. ICMP Type 3 Code 4 should be allowed from the Internet to your DMZ at least. Opening your Internal segmentation to this kind of traffic is questionable and depends on the facilities / activities / usage of the site and the level of filtering you wish to maintain.

If you will block incoming ICMP "Fragmentation Needed and Don't Fragment Bit was Set" your network performance will suffer from degradation. You should understand the security risks involving in opening this kind of traffic to your DMZ (& protected network) - The possibility of a Denial-of-Service, Inverse Mapping, Host Detection, and a one-way Covert communication channel (which was not been seen in the wild yet).

Another consideration could be the usage of network troubleshooting tools such as traceroute and ping. In the case of traceroute if the filtering device you are using does not support Stateful inspection with ICMP than allowing ICMP TTL Exceeded in Transit (Type 11, code 0) error messages inside the protected network could lead to various security hazards. The same goes with ping, where ICMP ECHO reply is even more dangerous when allowed inside the protected network (Inverse Mapping, Covert Channel and more security risks).

You can limit the number of systems that need to use the network troubleshooting tools with ACL, but bear in mind that those systems could be mapped from the Internet – and this is only the tip of the iceberg.

Internal Host(s) performance considerations – When blocking incoming ICMP Destination Unreachable Network/Host/Protocol/Port Unreachable ICMP error messages coming from the Internet, host(s) would hang when the destination system's network is unreachable/when a host is unreachable/when a protocol on the destination machine is not available/a port on a destination machine is closed. They all would hang until the timeout counter would reach zero. This little inconveniently is better than having the dangers other types of ICMP error messages inside your network can introduce.

Unless your filtering device is a real intelligence one, doing his work with dynamic tables and correlating correctly the ICMP replies with the requests, do not open your Internal network segment to no ICMP traffic type.

Some might offer to use a Proxy server with the ICMP protocol between the Internet and you protected network(s). A Proxy Server is only a tunnel – remember that.

[50] See Appendix B: "Fragmentation Needed but the Don't Fragment Bit was set" and the Path MTU Discovery Process.

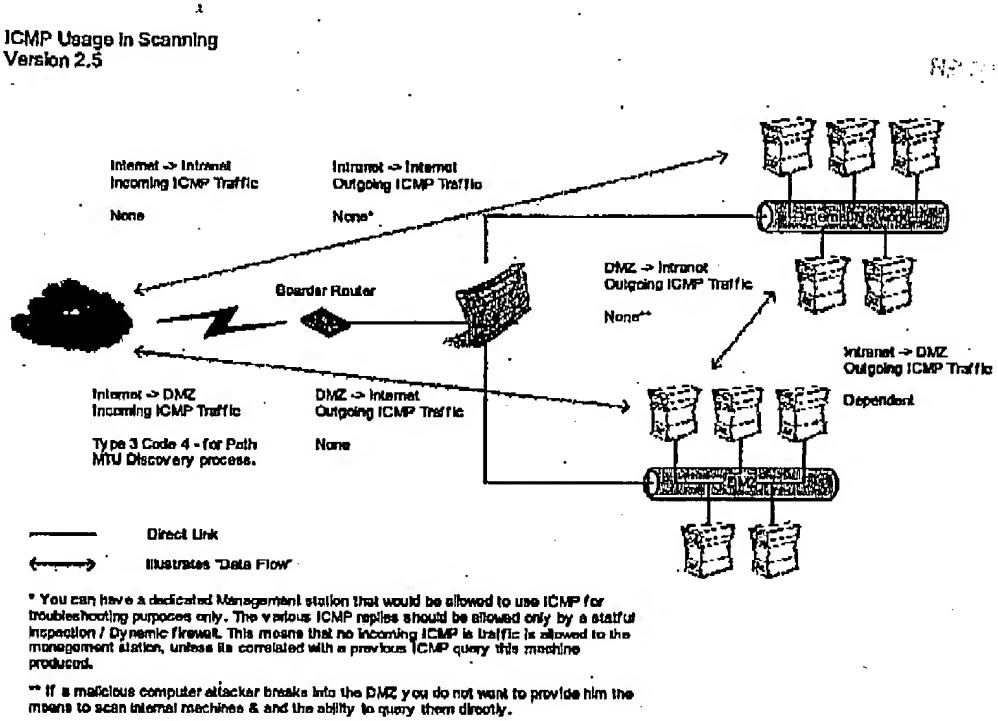

Figure 14: Firewall ICMP Filtering Rules

7.4 Other Problems – Why it is important to filter ICMP traffic in the Internal segmentation

Consider the following realistic scenario:

You have an Internal segment built with Microsoft based operating system machines (for the sake of the example only). A malicious computer attacker might send you a Trojan that will have Host detection and/or mapping capabilities. It will be hidden in an Email message (either as attachment or some other thing) a naïve user will open. After activation it will start to map internal hosts and internal segments and send the information back to the malicious computer attacker.

What will be the easiest method in order to map internal host(s)? Ping them.

How many of you reading this research have "management segments" that are allowed to use the Ping utility in order to verify that some Hosts are alive?

If something like this Trojan gets its way to this segment than probably your entire internal networking infrastructure (or the important part of it) will be revealed.

Some one might think that strong filtering or a good anti virus might help – forget it. I have seen people separating their work environment to more than two or three computers, but they always use the Email, and need to surf the web... (good ways to send the collected information out).

My suggestion is to configure internal host(s) not to answer for ICMP Query message types they should not answer for. I would restrict this to the maximum and not allow internal hosts to be queried with any ICMP Query message type.

Back to our monitoring problem - If you need to maintain management/monitoring capabilities, than I would suggest filtering the traffic in both ways from the management stations to the monitored systems in a way it would not be possible to simply query the last (dynamic filtering / stateful filtering with ICMP). Use a dedicated system for the querying and block the other machines in the management segment from doing so.

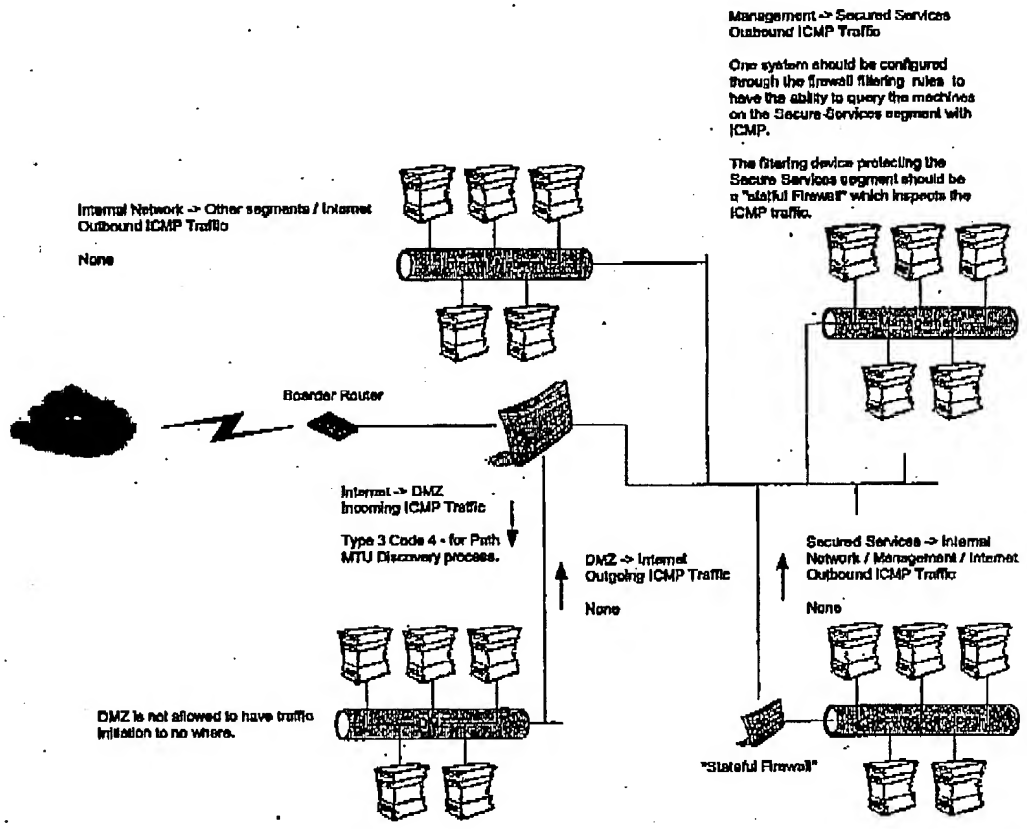

Figure 15: Internal segmentation ICMP Filtering Example

7.5 The Firewall

It is extremely important to block traffic, which is aimed at the Firewall itself. This rule will not block every thing. For example, ICMP error messages the firewall generates for various stimulus.

Some firewalls will hold a certain portion of a fragmented packet until the IP Header and the underlying protocol's header arrives. The ICMP error message for Fragment Reassembly Time Exceeded will not be of the Host, it will be of the Firewall spoofing it. Some Firewalls has the ability to spoof ICMP Echo Replies for Hosts they are defending. We will have the opportunity to fingerprint the operating system, which the firewall software is installed on.

We will gain an extremely important ability. Therefore it is recommended to have two basic rules when you configure your firewall's rule base. The first is to deny any traffic destined to the firewall

and the second would be to deny any error messages (or other conditions such as TCP reject etc.) that might help a malicious computer attacker in his task to fingerprint the Firewall itself.

8.0 Conclusion

The ICMP protocol is a very powerful tool in the hands of smart malicious computer attackers. Mapping, detecting, and fingerprinting of hosts and networking devices can be done in various ways as I have outlined in this paper.

It is extremely important to understand that ICMP traffic can be used for other malicious activities other than scanning, such as:

- Denial of Service Attacks
- Distributed Denial of Service Attacks
- Covert Channel Communications

Therefore filtering Inbound and Outbound ICMP traffic is very important and may help you in preventing risks to your computing environment.

9.0 Acknowledgment

9.1 Acknowledgment for version 1.0

I would like to thank the following people for their help with/during this research.

Ariel Pisetsky for going over this paper correcting my English, and for his moral support.

Christopher Tresco, Systems Administrator at the Massachusetts Institute of Technology provided necessary test systems to verify my findings.

Special thanks to mr2940 for his patience while I introduced my new ideas.

James Cudney, Michael, Pat, for their support when the times where bad.

9.1 Acknowledgment for version 2.0

I would like to thank Alfredo Andres Omella author of SING for his help.

I would like to thank Fyodor for his help providing me with necessary test systems.

I would like to thank the people who provided feedback to the first version of this research paper, and to the people who provided feedback to my Bugtraq posts.

9.2 Acknowledgment for version 2.5

I would like to thank Alfredo Andres Omella author of SING, for implementing some of the ideas I had into his tool.

I would like to thank Fyodor for his help providing me with necessary test systems.

Christopher Tresco, Systems Administrator at the Massachusetts Institute of Technology provided necessary test systems to verify my findings.

I would like to thank Simple Nomad for his support.

I would like to thank the huge amount of people who provided feedback for my work.

Appendix A: The ICMP Protocol[51]

Internet Control Message Protocol (ICMP) is used for two types of operations: when a *router* or a *destination host* need to inform the source host about errors in a datagram processing, and for probing the network with request messages in order to determine general characteristics about the network (getting the information back, hopefully, with the reply messages).

Some of ICMP's characteristics are:

- ICMP uses IP as if it were a higher-level protocol, however, ICMP is already an internal part of IP, and must be implemented by every IP module.
- ICMP is used to provide feedback about some errors in a datagram processing, not to make IP reliable. Datagrams may still be undelivered without any report of their loss. If a higher level protocol that use IP need reliability he must implement it.
- No ICMP messages are sent in response to ICMP messages to avoid infinite repetitions. The exception is a response to ICMP query messages (ICMP Types 0,8-10,13-18. See Table 1 ICMP Query Messages).
- For fragmented IP datagrams ICMP messages are only sent about errors on fragment zero (first fragment).
- ICMP error messages are never sent in response to a datagram that is *destined* to a *broadcast* or a *multicast* address.
- ICMP error messages are never sent in response to a datagram sent as a link layer broadcast.
- ICMP error messages are never sent in response to a datagram whose source address does not represents a unique host – the source IP address cannot be *zero*, a *loopback* address, a *broadcast* address or a *multicast* address.
- ICMP Error messages are never sent in response to an IGMP massage of any kind.
- When an ICMP message of ***unknown type*** is received, it must be silently *discarded*.
- Routers will almost always generate ICMP messages but when it comes to a destination host(s), the number of ICMP messages generated is implementation dependent.

ICMP Query Messages	ICMP error Messages
Echo	Destination Unreachable
Router Advertisement	Source Quench
Router Solicitation	Redirect
Time Stamp	Time Exceeded
Information	Parameter Problem
Address Mask	

Table 16: ICMP message types

[51] ICMP is described in RFC 972 (http://www.ietf.org/rfc/rfc0972.txt) with updates in: RFC 896 (Source Quench), RFC 950 (Address Mask Extensions), RFC 1191 (Path MTU Discovery) & RFC 1256 (Router Discovery). Further clarifications about the ICMP protocol are included in RFC 1122 and in RFC 1812. STD 2 has redefine and clarified much of ICMP's core functionality.

A.1 ICMP Messages

ICMP messages are sent in IP datagrams. The protocol number will be always one (ICMP), and the Type-of-Service will be zero. The IP data field will contain the actual ICMP message:

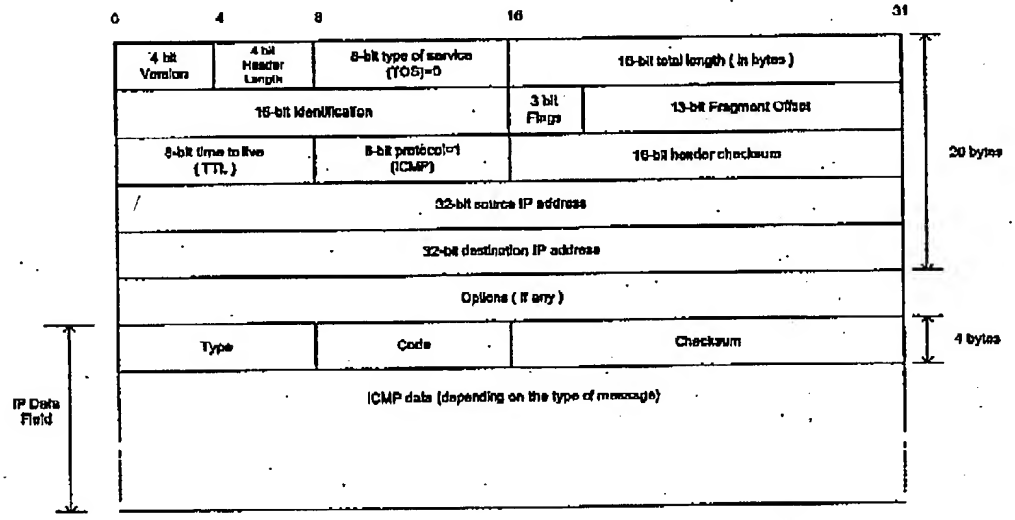

Figure 16: ICMP Message Format

ICMP error message length

Every ICMP error message includes the Internet (IP) Header and *at least* the first 8 data octets (bytes) of the datagram that triggered the error; more than 8 octets (bytes) *may* be sent; this header and data must be unchanged from the received datagram.

The **TYPE** field specifies the type of the message, while the error code for the datagram reported on by this ICMP message is contained in the **CODE** field. The code interpretation is dependent upon the message type.

Type	Name	Code
0	Echo Reply	0 No Code
1	Unassigned	
2	Unassigned	
3	Destination Unreachable[52]	0 Net Unreachable 1 Host Unreachable 2 Protocol Unreachable 3 Port Unreachable 4 Fragmentation Needed and Don't Fragment was Set 5 Source Route Failed 6 Destination Network Unknown 7 Destination Host Unknown 8 Source Host Isolated[53] 9 Communication with Destination Network is Administratively Prohibited[54] 10 Communication with Destination Host is Administratively Prohibited[55] 11 Destination Network Unreachable for Type of Service. 12 Destination Host Unreachable for Type of Service. 13 Communication Administratively Prohibited. 14 Host Precedence Violation 15 Precedence cutoff in effect
4	Source Quench	0 No Code
5	Redirect	0 Redirect Datagram for the Network (or subnet) 1 Redirect Datagram for the Host 2 Redirect Datagram for the Type of Service and Network 3 Redirect Datagram for the Type of Service and Host
6	Alternate Host Address	0 Alternate Address for Host
7	Unassigned	
8	Echo Request	0 No Code
9	Router Advertisement	0 No Code
10	Router Selection	0 No Code
11	Time Exceeded	0 Time to Live exceeded in Transit 1 Fragment Reassembly Time Exceeded
12	Parameter Problem	0 Pointer indicates the error 1 Missing a Required Option 2 Bad Length
13	Timestamp	0 No Code
14	Timestamp Reply	0 No Code

[52] RFC 972 defines codes 1-5. RFC 1122 defines codes 6-12. RFC 1812 defines codes 13-15.
[53] Reserved for use by U.S. military agencies.
[54] Reserved for use by U.S. military agencies.
[55] Reserved for use by U.S. military agencies.

Type	Name	Code
15	Information Request	0 No Code
16	Information Reply	0 No Code
17	Address Mask Request	0 No Code
18	Address Mask Reply	0 No Code
19	Reserved (for Security)	0 No Code
	20-29 reserved (for Robustness Experiment)	
30	Traceroute	
31	Datagram Conversion Error	
32	Mobile Host Redirect	
33	IPv6 Where-Are-You	
34	IPv6 I-Am-Here	
35	Mobile Registration Request	
36	Mobile Registration Reply	
39	SKIP	
40	Photuris	0 Reserved 1 unknown security parameters index 2 valid security parameters, but authentication failed 3 valid security parameters, but decryption failed

Table 17: ICMP Types & Codes

Checksum – contains the 16bit one's complement of the one's complement sum of the ICMP message starting with the ICMP Type field. For computing this checksum, the checksum field is assumed to be zero.

Data

- With ICMP error messages it will contain a part of the original IP message for which this ICMP message was generated. The length of the DATA field equals the IP datagram length less the IP header length. Every ICMP error message includes the Internet (IP) Header and *at least* the first 8 data octets (bytes) of the datagram that triggered the error; more than 8 octets (bytes) *may* be sent; this header and data must be unchanged from the received datagram.
- With ICMP query messages the Data field will contain dependent information upon the query type.

A.1 ICMP Error Messages

ICMP error messages are used to report a problem that prevented delivery. The nature of the problem should be a non-transient delivery problem.

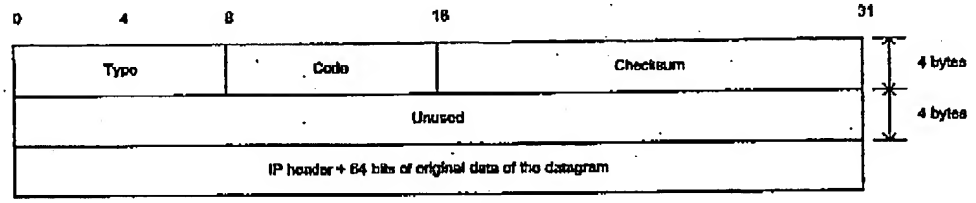

Figure 17: ICMP Error Message General Format

ICMP error message length
Every ICMP error message includes the IP Header (20 to 60 bytes) and *at least* the first 8 data bytes of the datagram that triggered the error; more than 8 bytes *may* be sent; this header and data must be unchanged from the received datagram.

AN ICMP error message length should be between 36 to 72 bytes.

The ICMP Protocol Rules for ICMP Error Messages

- ICMP Error messages are not sent for another ICMP Error message to prevent infinite loops.
- ICMP error messages are never sent in response to a datagram that is *destined* to a *broadcast* or a *multicast* address.
- ICMP error messages are never sent in response to a datagram sent as a link layer broadcast.
- ICMP error messages are never sent in response to a datagram whose source address does not represents a unique host – the source IP address cannot be *zero*, a *loopback* address, a *broadcast* address or a *multicast* address.
- ICMP Error messages are never sent in response to an IGMP massage of any kind.

A.1.1 ICMP Error Messages[56]

- Destination Unreachable (Type 3)
- Source Quench (Type 4)
- Redirect (Type 5)
- Time Exceeded (Type 11)
- Parameter Problem (Type 12)

A.1.1.1 Destination Unreachable (Type 3)
ICMP Destination Unreachable message type issued by a Destination Host:
A *destination host* issues a destination unreachable message when the protocol specified in the *protocol number* field of the original datagram is not active on the destination host, or the *specified port* is inactive.

ICMP Destination Unreachable message type issued by a Router:

[56] Some of the wording in this section are direct quotes from RFC 792 available from http://www.ietf.org/rfc/rfc0972.txt.

A *router* issue a destination unreachable message in response to a packet that it cannot forward because the destination (or next hop) is unreachable or a service is unavailable.

Code	Meaning	Explanation
0	Network Unreachable	Generated by a router if a route to the destination network is not available.
1	Host Unreachable	Generated by a router if a route to the destination host on a directly connected network is not available (does not respond to ARP).
2	Protocol Unreachable	Generated if the transport protocol designated in a datagram is not supported in the transport layer of the final destination.
3	Port Unreachable	Generated if the designated transport protocol (e.g. UDP) is unable to demultiplex the datagram in the transport layer of the final destination but has no protocol mechanism to inform the sender.
4	Fragmentation needed and DF flag Set	Generated if a router needs to fragment but cannot since the DF flag is set.
5	Source Route Failed	Generated if a router cannot forward a packet to the next hop in a source route option.
6	Destination Network Unknown	According to RFC 1812 this code should not be generated since it would imply on the part of the router that the destination network does not exist (net unreachable code 0 should be used instead of code 6).
7	Destination Host Unknown	Generated only when a router can determine (from link layer advice) that the destination host does not exist.
8	Source Host Isolated	Generated by a Router if it have been configured not to forward packets from source.
9	Communication with Destination Network is Administratively Prohibited	Generated by a Router if it has been configured to block access to the desired destination network.
10	Communication with Destination Host is Administratively Prohibited	Generated by a Router if it has been configured to block access to the desired destination host.
11	Network Unreachable for Type of Service	Generated by a router if a route to the destination network with the requested or default TOS is not available.
12	Host Unreachable for Type of Service	Generated if a router cannot forward a packet because its route(s) to the destination do not match either the TOS requested in the datagram or the default TOS (0).
13*	Communication Administratively Prohibited	Generated if a router cannot forward a packet due to administrative filtering (ICMP sender is not available at this time).
14	Host Precedence Violation	Sent by the first hop router to a host to indicate that a requested precedence is not permitted for the particular combination of source/destination host or network, upper layer protocol, and source/destination port.

Code	Meaning	Explanation
15	Precedence cutoff in effect	The network operators have imposed a minimum level of precedence required for operation, the datagram was sent with precedence below this level.

* Routers *may* have a configuration option that causes code 13 messages not to be generated. When this option is enabled, no ICMP error message is sent in response to a packet that is dropped because it's forwarding is administratively prohibited. Same is with type 14 & 15.

Table 18: Destination Unreachable Codes (Router)

The only type of ICMP Destination Unreachable error message, which is slightly different from the other, is Type 3 Code 4 – Fragmentation Needed but the Don't Fragment Bit was set.

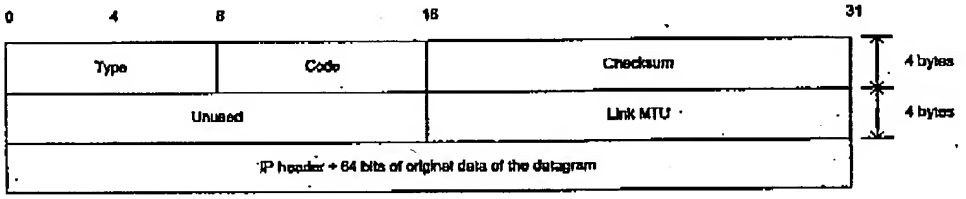

Figure 18: ICMP Fragmentation Needed but the Don't Fragment Bit was set Message Format

The Unused field will be 16 bits in length, instead of 32 bits, with this type of message. The rest of the 16 bits will be used to carry the MTU used for the link that could not deliver the datagram to the next hop (or destination) because the size of the datagram was too big to carry. Since this datagram could not be fragmented (the DF Bit was set) an error message has been sent to the sender indicating that a lower MTU should be used, hinting the size of the next hops links.

A.1.1.2 Source Quench (Type 4)

ICMP Source Quench message type Issued by a Router:
If a *router* sends this message, it means that the router does not have the buffer space needed to queue the datagrams for output to the next network on the route to the destination network.

RFC 1812 specify that a router *should not* generate Source Quench messages, but a router that does originate Source Quench message *must* be able to limit the rate at which they are generated (because it consumes bandwidth and it is an ineffective antidote to congestion).

A router receiving an ICMP Source Quench message type:
When a router receives such a message it *may* ignore it.

ICMP Source Quench message type Issued by a Host:
If a *destination host* sends this message (it *may* be implemented), it means that the datagrams arrive too fast to be processed. The ICMP source quench message is a request to the host to cut back the rate, which it is sending traffic to the Internet destination.

The ICMP header code would be always zero.

Host receiving an ICMP Source Quench message type:
An ICMP Source Quench message *must* be reported to the transport layer, UDP or TCP, the host should throttle itself back for a period of time, than gradually increase the transmission rate again.

A.1.1.3 Redirect (Type 5)

ICMP Redirect message type issued by a Router:
If a router generates this message, it means the host should send future datagrams for the network to the router who's IP is given in the ICMP message. The router should be always on the same subnet as the host who sent the datagram and the router that generated the ICMP redirect message.

A routing loop is generated when the router IP address matches the source IP address in the original datagram header.

Routers *must not* generate a Redirect Message unless *all* the following conditions are met:

- The packet is being forwarded out the same physical interface that it was received from,
- The IP source address in the packet is on the same Logical IP (Sub) network as the next-hop IP address, and
- The packet does not contain an IP source route option.

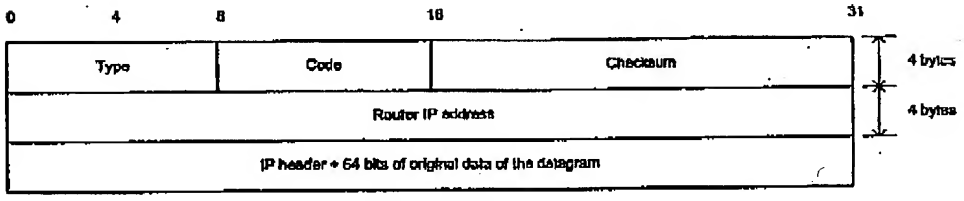

Figure 19: ICMP Redirect Message Format

A router receiving an ICMP Redirect message type:
A router *may* ignore ICMP Redirects when choosing a path for a packet originated by the router if the router is running a routing protocol or if forwarding is enabled on the router and on the interface over which the packet is being sent.

Four different codes can appear in the code field:

Code	Meaning
0	Redirect Datagram for the Network (or subnet)
1	Redirect Datagram for the Host

Code	Meaning
2	Redirect Datagram for the Type of Service and Network
3	Redirect Datagram for the Type of Service and Host

Table 19: Redirect Codes

ICMP Redirect message type issued by a Host:
A *host should not* send an ICMP Redirect message. Redirects are to be sent only by routers.

Host receiving an ICMP Redirect message type:
A *host* receiving a Redirect message *must* update its routing information accordingly. Every host *must* be prepared to accept both Host and Network Redirects.

The Redirect message *should* be silently discarded with the following cases:

- The new gateway address it specifies is not on the same connected (sub-) net through which the Redirect arrived.
- If the source of the Redirect is not the current first-hop gateway for the specified destination.

A.1.1.4 Time Exceeded (Type 11)

ICMP Time Exceeded message type Issued by a Router:
If a *router* discovers that the Time-To-Live field in an IP header of a datagram he process equals zero he will discard the datagram and generate an ICMP Time Exceeded Code 0 – transit TTL expired (this can also be an indicator of a routing loop problem).

When the router reassemble a packet that is destined for the router, it is acting as an Internet host. Host rules apply also when the router *receives* a Time Exceeded message.

A router *must* generate an ICMP Time Exceeded message code 0 when it discards a packet due to an expired TTL field. A router *may* have a per-interface option to disable origination of these messages on that interface, but that option *must* default to allowing the messages to be originated.

ICMP Time Exceeded message type issued by a Host:
If a *host* cannot reassemble a fragmented datagram due to missing fragments within its time limit it will discard the datagram and generate an ICMP Time Exceeded Code 1 – reassembly TTL. Exceeded.

A.1.1.5 Parameter Problem (Type 12)

ICMP Parameter Problem message is sent when a router (*must* generate this message) or a host (*should* generate this message) process a datagram and finds a problem with the IP header parameters. It is only sent if the error caused the datagram to be discarded.

The Parameter Problem message is generated usually for any error not specifically covered by another ICMP message.

If code 0 is used, the pointer field will point to the exact byte in the original IP Header, which caused the problem.

Three different codes can appear in the code field:

Codes	Meaning
0	Pointer indicates the error (unspecified error)
1	Missing a Required Option
2	Bad Length

Table 20: Parameter Problem Codes

Receipt of a parameter problem message generally indicates some local or remote implementation error.

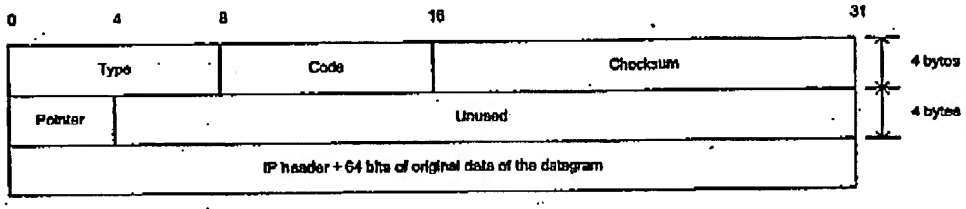

Figure 20: ICMP Parameter Problem Message Format

Appendix B: ICMP "Fragmentation Needed but the Don't Fragment Bit was set" and the Path MTU Discovery Process [57]

When one host needs to send data to another host, the data is transmitted in a series of IP datagrams. We wish the datagrams be the largest size possible that does not require fragmentation[58] along the path from the source host to the destination host.

Fragmentation by the IP layer raises few problems:

- If one fragment from a packet is dropped, we need to retransmit the whole packet.
- Load on the routers, which needs to do the fragmentation.
- Some simpler firewalls would block all fragments because they do not contain the header information for a higher layer protocol needed for filtering.

The Maximum Transfer Unit (***MTU***) is a link layer restriction on the maximum number of bytes of data in a single transmission. The smallest MTU of any link on the current path between two hosts is called the ***Path MTU***.

B.1 The PATH MTU Discovery Process

We use the Don't Fragment Bit Flag in the IP header to dynamically discover the Path MTU of a given route: The source host assumes that the PMTU of a path is the known MTU of its first hop. He will send all datagrams with that size, and set the Don't Fragment Bit. If along the path to the destination host, there is a router that needs to fragment the datagram in order to pass it to the next hop, an ICMP error message (Type 3 Code 4 "Fragmentation Needed and DF set") will be generated, since the Don't Fragment bit was set. When the sending host receives the ICMP error message he should reduce his assumed PMTU for the path.

The process can end when the estimated PMTU is low enough for the datagrams not to be fragmented. The source host itself can stop the process if he is willing to have the datagrams fragmented in some circumstances.

Usually the DF bit would be set in all datagrams, so if a route changes to the destination host, and the PMTU is lowered, than we would discover it.

The PMTU of a path might be increased over time, again because of a change in the routing topology. To detect it, a host should periodically increase its assumed PMTU for that link.

The link MTU field in the ICMP "Fragmentation Needed and DF set" error message, carries the MTU of the constricting hop, enabling the source host to know the exact value he needs to set the PMTU for that path to allow the voyage of the datagrams beyond that point (router) without fragmentation.

B.2 Host specification

A host must reduce his estimated PMTU for the relevant path when he receives the ICMP "Fragmentation Needed and the DF bit was set" error message. RFC 1191 does not outline a specific behavior that is expected from the sending host, because different applications may have different requirements, and different implementation architectures may favor different strategies.

---

[57] RFC 1191, http://www.ietf.org/rfc/rfc1191.txt, J. Mogul, S. Deering.

[58] When we send a packet that it is too large to be sent across a link as a single unit, a router needs to slice/split the packet into smaller parts, which contain enough information for the receiver to reassemble them. This is called fragmentation.

The only required behavior is that a host *must* attempt to avoid sending more messages with the same PMTU value in the near future. A host can either cease setting the Don't Fragment bit in the IP header (and allow fragmentation by the routers in the way) or reduce the datagram size. The better strategy would be to lower the message size because fragmentation will cause more traffic and consume more Internet resources.

A host using the PMTU Discovery process *must* detect decreases in Path MTU as fast as possible. A host *may* detect increases in Path MTU, by sending datagrams larger than the current estimated PMTU, which will usually be rejected by some router on the path to a destination since the PMTU usually will not increase. Since this would generate traffic back to the host, the check for the increases must be done at infrequent intervals. The RFC specify that an attempt for detecting an increasment *must not* be done less than 10 minutes after a datagram Too Big has been received for the given destination, or less than 2 minute after a previously successful attempt to increase.

The sending host must know how to handle an ICMP "Fragmentation Needed and the DF bit was set" error message that was sent by a device who does not know how to handle the PMTU protocol and does not include the next-hop MTU in the error message. Several strategies are available:

- The PMTU should be set to the minimum between the currently assumed PMTU and 576[59]. The DF bit should not be set in future datagrams for that path.
- Searching for the accurate value for the PMTU for a path. We keep sending datagrams with the DF bit set with lowered PMTU until we do not receive errors.

A host must not reduce the estimation of a Path MTU value below 68 bytes.

A host MUST not increase its estimate of the Path MTU in response to the contents of a Datagram Too Big message.

B.3 Router Specification

When a router cannot forward a datagram because it exceeded the MTU of the next-hop network and the Don't Fragment bit was set, he is required to generate an ICMP Destination Unreachable message to the source of the datagram., with the appropriate code indicating "Fragmentation needed and the Don't Fragment Bit was set". In the error message the router *must* include the MTU of the next-hop in a 16bit field inside the error message.

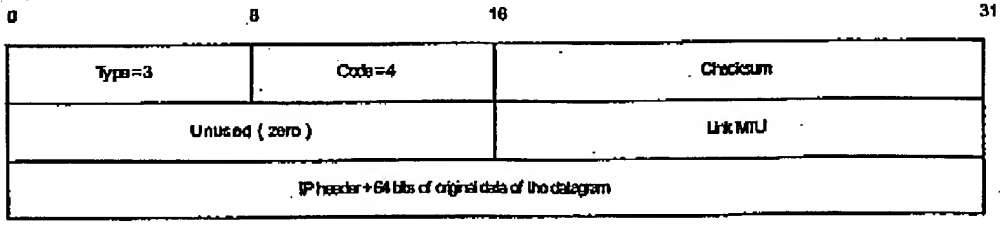

Figure 21: ICMP Fragmentation Required with Link MTU

[59] The usage of the lesser between 576 and the first-hop MTU as the PMTU for a destination, which is not connected to the same network was the old implementation. The results were the use of smaller datagrams than necessary, waste of Internet resources, and not being optimal.

The value of the next-hop MTU field should be set to the size in bytes of the largest datagram that could be forwarded, along the path of the original datagram, without being fragmented by this router. The size includes IP header plus IP data and no lower level headers should be included.

Because every router should be able to forward a datagram of 68 bytes without fragmenting it, the link MTU field should not contain a value less than 68.

B.4 The TCP MSS (Maximum Segment Size) Option and PATH MTU Discovery Process

The RFC specify that a host that is doing Path MTU Discovery *must not* send datagrams larger than 576 bytes unless the receiving host grants him permission.

When we are establishing a TCP connection both sides announce the maximum amount of data in one packet that should be sent by the remote system – The maximum segment size, MSS (if one of the ends does not specify an MSS, it defaults to 536 – there is no permission from the other end to send more than this amount). The packet generated would be, normally, 40 bytes larger than the MSS; 20 bytes for the IP header and 20 bytes for the TCP header. Most systems announce an MSS that is determined from the MTU on the interface that the traffic to the remote system passes out from the system through.

Each side upon receiving the MSS of the other side should not send any segments larger than the MSS received, regardless of the PMTU. After receiving the MSS value the Path MTU Discovery process will start to take affect. We will send our IP packets with the DF bit set allowing us to recognize points in the path to our destination that cannot process packets larger as the MSS of the destination host plus 40 bytes. When such an ICMP error message arrives, we should lower the PMTU to a path (according to the link MTU field, or if not used, to use the rules regarding the old implementation) and retransmit. The value of the link MTU cannot be higher than the MSS of the destination host. When retransmission occurs resulting from ICMP type 3 code 4 error message, the congestion windows should not change, but slow start should be initiated. The process continues until we adjust the correct PMTU of a path (not receiving ICMP error messages from the intermediate routers) which will allow us to fragment at the TCP layer which is much more efficient than at the IP layer.

Appendix C: Mapping Operating Systems for answering/discarding ICMP query message types

Operating System	Info: Request	Time Stamp Request	Address Mask Request	Address Mask Request Frag.	IP TTL on ICMP datagrams - In Reply -	IP TTL on ICMP datagrams - In Req. -
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2 (*)	-	+	-	-	255	64
Redhat LINUX 6.2 Kernel 2.2.14 (*)	-	+	-	-	255	64
LINUX Kernel 2.0.x					64	64
FreeBSD 4.0 (*)	-	+	-	-	255	255
FreeBSD 3.4	-	+	-	-	255	255
OpenBSD 2.7	-	+	-	-	255	255
OpenBSD 2.6	-	+	-	-	255	255
NetBSD	-	+	-	-	255	
BSDI BSD/OS 4.0	-	+	-	-	255	
BSDI BSD/OS 3.1	-	+	-	-	255	
Solaris 2.5.1	-	+	+	+ (0.0.0.0)	255	255
Solaris 2.6	-	+	+	+ (0.0.0.0)	255	255
Solaris 2.7 (*)	-	+	+	+ (0.0.0.0)	255	255
Solaris 2.8	-	+	+	+ (0.0.0.0)	255	255
HP-UX v10.20	+	+	-	-	255	255
HP-UX v11.0	-	-	+	+ (0.0.0.0)	255	
Compaq Tru64 v5.0 (*)	+	+	-	-	64	
Irix 6.5.3 (*)	-	+	-	-	255	
Irix 6.5.8 (*)	-	+	-	-	255	
AIX 4.1 (*)	+	+	-	-	255	
AIX 3.2 (*)	+	+	-	-	255	
ULTRIX 4.2 – 4.5 (*)	+	+	+	+	255	
OpenVMS v7.1-2 (*)	+	+	+	+	255	
Novell Netware 5.1 SP1 (*)	-	-	-	-	128	
Novell Netware 5.0 (*)	-	-	-	-	128	
Novell Netware 3.12	-	-	-	-	128	
Windows 95	-	-	+	+	32	32
Windows 98 (*)	-	+	+	+	128	32
Windows 98 SE (*)	-	+	+	+	128	32
Windows ME (*)	-	+	-	-	128	32
Windows NT 4 WRKS SP 3 (*)	-	-	+	+	128	32
Windows NT 4 WRKS SP 6a (*)	-	-	-	-	128	32
Windows NT 4 Server SP4	-	-	-	-	128	32
Windows 2000 Professional (*)	-	+	-	-	128	128
Windows 2000 Server (*)	-	+	-	-	128	128

ICMP Usage in Scanning
Version 2.5

Networking Devices	Info. Request	Time Stamp Request	Address Mask Request	Address Mask Request Frag.	IP TTL on ICMP datagrams - In Reply -	IP TTL on ICMP datagrams - In Req. -
Cisco Catalyst 5505 with OSS v4.5	+	+	+	-	60	60
Cisco Catalyst 2900XL with IOS 11.2	+	+	-	-	255	
Cisco 3600 with IOS 11.2	+	+	-	-	255	
Cisco 7200 with IOS 11.3	+	+	-	-	255	255
Intel Express 8100 ISDN Router (*)	-	-	+	+	64	

Appendix D: ICMP Query Message Types with Code field !=0

Operating System	Info. Request	Time Stamp Request	Address Mask Request		ECHO Request	
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2 (*)	-	+ (0)	-		+ (!=0)	
Redhat LINUX 6.2 Kernel 2.2.14 (*)	-	+ (0)	-		+ (!=0)	
FreeBSD 4.0 (*)	-	+ (!=0)	-		+ (!=0)	
FreeBSD 3.4	-	+ (!=0)	-			
OpenBSD 2.7	-	+ (!=0)	-		+ (!=0)	
OpenBSD 2.6	-	+ (!=0)	-		+ (!=0)	
NetBSD	-	+ (!=0)	-		+ (!=0)	
BSDI BSD/OS 4.0 (*)	-	+ (!=0)	-		+ (!=0)	
BSDI BSD/OS 3.1 (*)	-	+ (!=0)	-		+ (!=0)	
Solaris 2.5.1	*	+ (!=0)	+ (!=0)		+ (!=0)	
Solaris 2.6	*	+ (!=0)	+ (!=0)		+ (!=0)	
Solaris 2.7 (*)	*	+ (!=0)	+ (!=0)		+ (!=0)	
Solaris 2.8	*	+ (!=0)	+ (!=0)		+ (!=0)	
HP-UX v10.20	+ (!=0)	+ (!=0)	-			
HP-UX v11.0	-	-	+ (!=0)		+ (!=0)	
Compaq Tru64 v5.0 (*)	+ (!=0)	+ (!=0)	-		+ (!=0)	
Irix 6.5.9 (*)	-	+ (!=0)	-		+ (!=0)	
Irix 6.5.8 (*)	-	+ (!=0)	-		+ (!=0)	
AIX 4.1 (*)	+ (!=0)	+ (!=0)	-		+ (!=0)	
Aix 3.2 (*)	+ (!=0)	+ (!=0)	-			
ULTRIX 4.2 - 4.5 (*)	+ (!=0)	+ (!=0)	+ (!=0)		+ (!=0)	
OpenVMS v7.1-2 (*)	+ (!=0)	+ (!=0)	+ (!=0)		+ (!=0)	
Novell Netware 5.1 SP1 (*)	-	-	-		+ (!=0)	
Novell Netware 5.0 (*)	-	-	-		+ (!=0)	
Novell Netware 3.12 (*)	-	-	-		+ (!=0)	
Windows 95	-	-	+		+ (0)	
Windows 98 (*)	-	- (CHANGE)	+		+ (0)	
Windows 98 SE (*)	-	- (CHANGE)	+		+ (0)	
Windows ME (*)	-	- (CHANGE)	-		+ (0)	
Windows NT 4 WRKS SP 3 (*)	-	-	+		+ (0)	
Windows NT 4 WRKS SP 6a (*)	-	-	-		+ (0)	
Windows NT 4 Server SP4	-	-	-		+ (0)	
Windows 2000 Professional (*)	-	- (CHANGE)	-		+ (0)	
Windows 2000 Server (*)	-	- (CHANGE)	-		+ (0)	

ICMP Usage in Scanning
Version 2.5

Networking Devices	Info. Request	Time Stamp Request	Address Mask Request		ECHO Request	
Cisco Catalyst 5505 with OSS v4.5	+	+	+		+ (10)	
Cisco Catalyst 2900XL with IOS 11.2	+	+	-		+ (10)	
Cisco 3600 with IOS 11.2					+ (10)	
Cisco 7200 with IOS 11.3	+	+	-		+ (10)	
Intel Express 8100 ISDN Router (*)						

Appendix E: ICMP Query Message Types aimed at a Broadcast Address

Operating System	Info. Request Broadcast	Time Stamp Request Broadcast	Address Mask Request Broadcast	Echo Request Broadcast
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2				+
Redhat LINUX 6.2 Kernel 2.2.14 (*)	-	+	-	+
FreeBSD 4.0 (*)	-	-	-	-
FreeBSD 3.4	-	-	-	-
OpenBSD 2.7	-	-	-	-
OpenBSD 2.6	-	-	-	-
NetBSD				
BSDI BSD/OS 4.0 (*)				
BSDI BSD/OS 3.1 (*)				
Solaris 2.5.1	*	+	-	+
Solaris 2.6	*	+	-	+
Solaris 2.7	*	+	-	+
Solaris 2.8	*	+	-	+
HP-UX v10.20	+	+	-	+
Compaq Tru64 v5.0 (*)				
Irix 6.5.3 (*)				
Irix 6.5.8 (*)				
AIX 4.1 (*)				
AIX 3.2 (*)				
ULTRIX 4.2 – 4.5 (*)				
OpenVMS v7.1-2 (*)				
Novell Netware 5.1 SP1 (*)				
Novell Netware 5.0 (*)				
Novell Netware 3.12 (*)				
Windows 95				
Windows 98	-	-	-	-
Windows 98 SE (*)	-	-	-	-
Windows ME (*)	-	-	-	-
Windows NT 4 WRKS SP 3 (*)	-	-	-	-
Windows NT 4 WRKS SP 6a (*)	-	-	-	-
Windows NT 4 Server SP4	-	-	-	-
Windows 2000 Professional (*)	-	-	-	-
Windows 2000 Server (*)	-	-	-	-

ICMP Usage in Scanning
Version 2.5

Networking Devices	Info. Request Broadcast	Time Stamp Request Broadcast	Address Mask Request Broadcast		Echo Broadcast	
Cisco Catalyst 5505 with OSS v4.5	+	+	+		+	
Cisco Catalyst 2900XL with IOS 11.2	+	-	-		+	
Cisco 3600 with IOS 11.2	+	-	-			
Cisco 7200 with IOS 11.3	+	-	-		+	
Intel Express 8100 ISDN Router (*)	-	-	-		-	Big Question Marks

Appendix F: Precedence Bits Echoing with ICMP Query Request & Reply

Operating System	Information Request With Precedence!=0	Time Stamp Request With Precedence!=0	Address Mask Request With Precedence!=0	Echo Request With Precedence!=0
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	Not Answering	!=0x00	Not Answering	!=0x00
Redhat LINUX 6.2 Kernel 2.2.14	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.0	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.1.1	Not Answering	!=0x00	Not Answering	
OpenBSD 2.7	Not Answering		Not Answering	!=0x00
OpenBSD 2.6	Not Answering		Not Answering	!=0x00
NetBSD	Not Answering		Not Answering	!=0x00
BSDI BSD/OS 4.0	Not Answering		Not Answering	!=0x00
BSDI BSD/OS 3.1	Not Answering		Not Answering	!=0x00
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.7	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.8	Not Implemented	!=0x00	!=0x00	!=0x00
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	!=0x00 -> 0x00	!=0x00 -> 0x00
Compaq Tru64 v5.0	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.3	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.2.1	!=0x00	!=0x00	Not Answering	!=0x00
AIX 4.1	!=0x00	!=0x00	Not Answering	!=0x00
AIX 3.2	!=0x00	!=0x00	Not Answering	!=0x00
ULTRIX 4.2 – 4.5	0x00	0x00	0x00	0x00
OpenVMS v7.1-2	0x00	0x00	0x00	!=0x00
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering			!=0x00
Windows 98 SE	Not Answering	0x00	0x00	!=0x00
Windows ME	Not Answering	0x00	Not Answering	!=0x00
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		!=0x00
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	!=0x00
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	!=0x00
Windows 2000 Professional	Not Answering	0x00	Not Answering	0x00
Windows 2000 Server	Not Answering	0x00	Not Answering	0x00

Appendix G: ICMP Query Message Types with TOS! = 0

Operating System	Information Request With TOS!=0x00	Time Stamp Request With TOS!=0x00	Address Mask Request With TOS!=0x00	Echo Request With TOS!=0x00
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2 (*)	Not Answering	!=0x00	Not Answering	!=0x00
Redhat LINUX 6.2 Kernel 2.2.14 (*)	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 4.0 (*)	Not Answering	!=0x00	Not Answering	!=0x00
FreeBSD 3.4	Not Answering		Not Answering	
OpenBSD 2.7 (*)	Not Answering	!=0x00	Not Answering	!=0x00
OpenBSD 2.6	Not Answering	!=0x00	Not Answering	!=0x00
NetBSD	Not Answering	!=0x00	Not Answering	!=0x00
BSDI BSD/OS 4.0 (*)	Not Answering	!=0x00	Not Answering	!=0x00
BSDI BSD/OS 3.1 (*)	Not Answering	!=0x00	Not Answering	!=0x00
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented			
Solaris 2.7 (*)	Not Implemented	!=0x00	!=0x00	!=0x00
Solaris 2.8 (*)	Not Implemented	!=0x00	!=0x00	!=0x00
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	!=0x00	!=0x00
Compaq Tru64 v5.0 (*)		!=0x00	Not Answering	!=0x00
Irix 6.5.3 (*)	Not Answering	!=0x00	Not Answering	!=0x00
Irix 6.5.8 (*)	Not Answering	!=0x00	Not Answering	!=0x00
AIX 4.1 (*)		!=0x00	Not Answering	!=0x00
AIX 3.2 (*)		!=0x00	Not Answering	!=0x00
ULTRIX 4.2 – 4.5 (*)		0x00	0x00	0x00
OpenVMS v7.1-2 (*)		!=0x00	!=0x00	!=0x00
Novell Netware 5.1 SP1 (*)	Not Answering	Not Answering	Not Answering	0x00
Novell Netware 5.0 (*)	Not Answering	Not Answering	Not Answering	0x00
Novell Netware 3.12 (*)	Not Answering	Not Answering	Not Answering	0x00
Windows 95	Not Answering	Not Answering		
Windows 98 (*)	Not Answering	0x00	0x00	!=0x00
Windows 98 SE (*)	Not Answering	0x00		!=0x00
Windows ME (*)	Not Answering	0x00	Not Answering	!=0x00
Windows NT 4 WRKS SP 3 (*)	Not Answering	Not Answering		!=0x00
Windows NT 4 WRKS SP 6a (*)	Not Answering	Not Answering	Not Answering	!=0x00
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	!=0x00
Windows 2000 Professional (*)	Not Answering	0x00	Not Answering	0x00
Windows 2000 Server (*)	Not Answering	0x00	Not Answering	0x00

Appendix H: Echoing the TOS Byte Unused bit

Operating System	Information Request With Unused=1	Time Stamp Request With Unused=1	Address Mask Request With Unused=1	Echo Request With Unused=1
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	Not Answering	0x1	Not Answering	0x1
Redhat LINUX 6.2 Kernel 2.2.14	Not Answering	0x1	Not Answering	0x1
FreeBSD 4.0	Not Answering	0x1	Not Answering	0x1
FreeBSD 4.1.1	Not Answering	0x1	Not Answering	0x1
OpenBSD 2.7	Not Answering		Not Answering	
OpenBSD 2.6	Not Answering		Not Answering	
NetBSD	Not Answering		Not Answering	
BSDI BSD/OS 4.0	Not Answering		Not Answering	
BSDI BSD/OS 3.1	Not Answering		Not Answering	
Solaris 2.5.1	Not Implemented			
Solaris 2.6	Not Implemented	0x1	0x1	0x1
Solaris 2.7	Not Implemented	0x1	0x1	0x1
Solaris 2.8	Not Implemented	0x1	0x1	0x1
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	0x1	0x1
Compaq Tru64 v5.0	0x1	0x1	Not Answering	0x1
AIX 4.3	0x1	0x1	Not Answering	0x1
AIX 4.2.1	0x1	0x1	Not Answering	0x1
AIX 4.1	0x1	0x1	Not Answering	0x1
AIX 3.2	0x1	0x1	Not Answering	0x1
ULTRIX 4.2 – 4.5	0x0	0x0	0x0	0x0
OpenVMS v7.1-2	0x1	0x1	0x1	0x1
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	0x0	0x0	0x1
Windows 98 SE	Not Answering	0x0	0x0	0x1
Windows ME	Not Answering	0x0	Not Answering	0x1
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	0x1
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	
Windows 2000 Professional	Not Answering	0x0	Not Answering	0x0
Windows 2000 Server	Not Answering	0x0	Not Answering	0x0

Appendix I: Using the Unused Bit

Operating System	Info. Request	Time Stamp Request	Address Mask Request	Echo Request
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2	Not Answering	-	Not Answering	-
Redhat LINUX 6.2 Kernel 2.2.14	Not Answering	-	Not Answering	-
FreeBSD 4.0	Not Answering	-	Not Answering	-
FreeBSD 3.4	Not Answering	-	Not Answering	-
OpenBSD 2.7	Not Answering	-	Not Answering	-
OpenBSD 2.6	Not Answering	-	Not answering	-
NetBSD	Not Answering	-	Not Answering	-
BSDI BSD/OS 4.0	Not Answering	-	Not Answering	-
BSDI BSD/OS 3.1	Not Answering	-	Not Answering	-
Solaris 2.5.1	Not Answering	+	+	+
Solaris 2.6	Not Answering	+	+	+
Solaris 2.7	Not Answering	+	+	+
Solaris 2.8	Not Answering	+	+	+
HP-UX v10.20	-	-	Not Answering	-
HP-UX v11.0	Not Answering	Not Answering	+	+
Compaq Tru64 v5.0	-	-	Not Answering	-
Irix 6.5.3	Not Answering	-	Not Answering	-
Irix 6.5.8	Not Answering	-	Not Answering	-
AIX 4.1	-	-	Not Answering	-
AIX 3.2	-	-	Not Answering	-
ULTRIX 4.2 – 4.5	-	-	-	-
OpenVMS v7.1-2	-	-	-	-
Novell Netware 5.1 SP1	Not Answering	Not Answering	Not Answering	-
Novell Netware 5.0	Not Answering	Not Answering	Not Answering	-
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	-
Windows 95	Not Answering	Not Answering		
Windows 98	Not Answering	-	-	-
Windows 98 SE	Not Answering	-	-	-
Windows ME	Not Answering		Not Answering	
Windows NT 4 WRKS SP 3	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a	Not Answering	Not Answering	Not Answering	-
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	-
Windows 2000 Professional	Not Answering	-	Not Answering	-
Windows 2000 Server	Not Answering	-	Not Answering	-

Appendix J: DF Bit Echoing

Operating System	Info. Request	Time Stamp Request	Address Mask Request	Echo Request
Debian GNU/ LINUX 2.2, Kernel 2.4 test 2 (*)	Not Answering	+ (- DF)	Not Answering	+ (- DF)
Redhat LINUX 6.2 Kernel 2.2.14 (*)	Not Answering	+ (- DF)	Not Answering	+ (- DF)
FreeBSD 4.0 (*)	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
FreeBSD 3.4	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
OpenBSD 2.7	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
OpenBSD 2.6	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
NetBSD	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
BSDI BSD/OS 4.0	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
BSDI BSD/OS 3.1	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Solaris 2.5.1	Not Answering			
Solaris 2.6	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
Solaris 2.7 (*)	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
Solaris 2.8	Not Answering	+ (+ DF)	+ (+ DF)	+ (+ DF)
HP-UX v10.20			Not Answering	
HP-UX v11.0	Not Answering	Not Answering	+ (+ DF)	+ (+ DF)
Compaq Tru64 v5.0 (*)		+ (+ DF)	Not Answering -	+ (+ DF)
Irix 6.5.3 (*)	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
Irix 6.5.8 (*)	Not Answering	+ (+ DF)	Not Answering	+ (+ DF)
AIX 4.1 (*)		+ (+ DF)	Not Answering	+ (+ DF)
AIX 3.2 (*)		+ (+ DF)	Not Answering	+ (+ DF)
ULTRIX 4.2 – 4.5 (*)		+ (- DF)	+ (- DF)	+ (- DF)
OpenVMS v7.1-2 (*)		+ (+ DF)	+ (+ DF)	+ (+ DF)
Novell Netware 5.1 SP1 (*)	Not Answering	Not Answering	Not Answering	+ (- DF)
Novell Netware 5.0 (*)	Not Answering	Not Answering	Not Answering	+ (- DF)
Novell Netware 3.12	Not Answering	Not Answering	Not Answering	+ (- DF)
Windows 95	Not Answering	Not Answering		
Windows 98 (*)	Not Answering	+ (- DF)	+ (- DF)	+ (+ DF)
Windows 98 SE (*)	Not Answering	+ (- DF)	+ (- DF)	+ (+ DF)
Windows ME (*)	Not Answering	+ (- DF)	Not Answering	+ (+ DF)
Windows NT 4 WRKS SP 3 (*)	Not Answering	Not Answering		
Windows NT 4 WRKS SP 6a (*)	Not Answering	Not Answering	Not Answering	+ (+ DF)
Windows NT 4 Server SP4	Not Answering	Not Answering	Not Answering	+ (+ DF)
Windows 2000 Professional (*)	Not Answering	+ (- DF)	Not Answering	+ (+ DF)
Windows 2000 Server (*)	Not Answering	+ (- DF)	Not Answering	+ (+ DF)

Appendix K: ICMP Error Message Echoing Integrity with ICMP Port Unreachable Error Message

Operating System	DF Bit set with the Reply?	IP Total Length	IP Identification	IP TTL field value	IP Header Checksum	UDP Checksum
LINUX Kernel 2.2.x	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
LINUX Kernel 2.4	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
FreeBSD 4.0	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
FreeBSD 4.11	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
BSDI 4.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed. Now equals to ZERO!	Same
Sun Solaris 2.6	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Sun Solaris 2.7	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Sun Solaris 2.8[60]	Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
HPUX 11.0	No -> Yes	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Compaq Tru64	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Changed. Now equal to ZERO!
DG-UX 5.6	No	Same	Same	Changed according to hop	Changed because of new parameters.	Changed. Now equal to ZERO!

[60] The DF Bit is set.

				count.		
AIX 4.3 fp2, 4.3, 4.2.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed because of new parameters.	Changed. Now equal to ZERO!
AIX 4.1	No	Changed (20 bytes more)	Same	Changed according to hop count	Changed because of new parameters.	Same
ULTRIX	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count	Changed. Now equals to ZERO!	Changed. Now equal to ZERO!
OpenVMS	No	Same	Changed. The first two bits are flipped with the second pair. Gives a new value.	Changed according to hop count	Changed. Now equals to ZERO!	Changed. Now equal to ZERO!
Microsoft windows 98 Mirosoft Windows 98SE	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows ME	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows NT 4	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same
Microsoft Windows 2000 Family	No	Same	Same	Changed according to hop count.	Changed because of new parameters.	Same

Appendix L: A Snort Rule Base for (more Advanced) Basic ICMP Traffic

The following generic ICMP basic Snort rule base is also available for download from: http://www.sys-security.com/archive/snort/icmp_rules/ICMP_basic_plus.

```
alert icmp any any -> any any (msg:"ICMP Echo Reply"; itype: 0; icode:
0;)
alert icmp any any -> any any (msg:"ICMP Echo Reply (Undefined Code!)";
itype: 0;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 1)"; itype:
1; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Tupe 1)
(Undefined Code)"; itype: 1;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 2)"; itype:
2; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 2)"
(Undefined Code); itype: 2;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Network Unreachable)"; itype: 3; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable (Host
Unreachable)"; itype: 3; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Protocol Unreachable)"; itype: 3; icode: 2;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable (Port
Unreachable)"; itype: 3; icode: 3;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Fragmentation Needed and DF bit was set)"; itype: 3; icode: 4;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Source Route Failed)"; itype: 3; icode: 5;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Destination Network Unknown)"; itype: 3; icode: 6;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Destination Host Unknown)"; itype: 3; icode: 7;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Source Host Isolated)"; itype: 3; icode: 8;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Communication with Destination Network is Administratively
Prohibited)"; itype: 3; icode: 9;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Communication with Destination Host is Administratively Prohibited)";
itype: 3; icode: 10;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Network Unreachable for Type of Service)"; itype: 3; icode: 11;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable (Host
Unreachable for Type of Service)"; itype: 3; icode: 12;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Communication Administratively Prohibited)"; itype: 3; icode: 13;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable (Host
Precedence Violation)"; itype: 3; icode: 14;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Precedence Cutoff in effect)"; itype: 3; icode: 15;)
alert icmp any any -> any any (msg:"ICMP Destination Unreachable
(Undefined Code!)"; itype: 3;)
alert icmp any any -> any any (msg:"ICMP Source Quench"; itype: 4;
icode: 0;)
```

```
alert icmp any any -> any any (msg:"ICMP Source Quench (Undefined
Code!)"; itype: 4;)
alert icmp any any -> any any (msg:"ICMP Redirect (for Network or
Subnet)"; itype: 5; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Redirect (for Host)"; itype:
5; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Redirect (for TOS and
Network)"; itype: 5; icode: 2;)
alert icmp any any -> any any (msg:"ICMP Redirect (for TOS and Host)";
itype: 5; icode: 3;)
alert icmp any any -> any any (msg:"ICMP Redirect (Undefined Code!)";
itype: 5;)
alert icmp any any -> any any (msg:"ICMP Alternate Host Address";
itype: 6; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Alternate Host Address
(Undefined Code!)"; itype: 6;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 7)"; itype:
7; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Unassigned! (Type 7)
(Undefined Code!)"; itype: 7;)
alert icmp any any -> any any (msg:"ICMP Echo Request"; itype: 8;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Echo Request (Undefined
Code!)"; itype: 8;)
alert icmp any any -> any any (msg:"ICMP Router Advertisment"; itype:
9; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Router Advertisment (Undefined
Code!)"; itype:9 ;)
alert icmp any any -> any any (msg:"ICMP Router Selection"; itype: 10;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Router Selection (Undefined
Code!)"; itype: 10;)
alert icmp any any -> any any (msg:"ICMP Time-To-Live Exceeded in
Transit"; itype: 11; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Fragment Reassembly Time
Exceeded"; itype: 11; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Time Exceeded (Undefined
Code!)"; itype: 11;)
alert icmp any any -> any any (msg:"ICMP Parameter Problem Code 0
(unspecified Error)"; itype: 12; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Parameter Problem Code 1
(Missing a Requiered Option)"; itype: 12; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Parameter Problem Code 2 (Bad
Length)"; itype: 12; icode: 2;)
alert icmp any any -> any any (msg:"ICMP Parameter Problem (Undefined
Code!)"; itype: 12;)
alert icmp any any -> any any (msg:"ICMP Timestamp Request"; itype: 13;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Timestamp Request (Undefined
Code!)"; itype: 13;)
alert icmp any any -> any any (msg:"ICMP Timestamp Reply"; itype: 14;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Timestamp Reply (Undefined
Code!)"; itype: 14;)
alert icmp any any -> any any (msg:"ICMP Information Request"; itype:
15; icode: 0;)
```

```
alert icmp any any -> any any (msg:"ICMP Information Request (Undefined
Code!)"; itype: 15;)
alert icmp any any -> any any (msg:"ICMP Information Reply"; itype: 16;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP Information Reply (Undefined
Code!)"; itype: 16;)
alert icmp any any -> any any (msg:"ICMP Address Mask Request"; itype:
17; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Address Mask Request
(Undefined Code!)"; itype: 17;)
alert icmp any any -> any any (msg:"ICMP Address Mask Reply"; itype:
18; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Address Mask Reply (Undefined
Code!)"; itype: 18;)
alert icmp any any -> any any (msg:"ICMP Reserved for Security (Type
19)"; itype: 19; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Reserved for Security (Type
19) (Undefined Code!)"; itype: 19;)
alert icmp any any -> any any (msg:"ICMP Traceroute"; itype: 30; icode:
0;)
alert icmp any any -> any any (msg:"ICMP Traceroute (Undefined Code!";
itype: 30;)
alert icmp any any -> any any (msg:"ICMP Datagram Conversion Error";
itype: 31; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Datagram Conversion Error
(Undefined Code!)"; itype: 31;)
alert icmp any any -> any any (msg:"ICMP Mobile Host Redirect"; itype:
32; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Mobile Host Redirect
(Undefined Code!)"; itype: 32;)
alert icmp any any -> any any (msg:"ICMP IPV6 Where-Are-You"; itype:
33; icode: 0;)
alert icmp any any -> any any (msg:"ICMP IPV6 Where-Are-You (Undefined
Code!)"; itype: 33;)
alert icmp any any -> any any (msg:"ICMP IPV6 I-Am-Here"; itype: 34;
icode: 0;)
alert icmp any any -> any any (msg:"ICMP IPV6 I-Am-Here (Undefined
Code!"; itype: 34;)
alert icmp any any -> any any (msg:"ICMP Mobile Registration Request";
itype: 35; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Mobile Registration Request
(Undefined Code!"; itype: 35;)
alert icmp any any -> any any (msg:"ICMP Mobile Registration Reply";
itype: 36; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Mobile Registration Reply
(Undefined Code!)"; itype: 36;)
alert icmp any any -> any any (msg:"ICMP SKIP"; itype: 39; icode: 0;)
alert icmp any any -> any any (msg:"ICMP SKIP (Undefined Code!"; itype:
39;)
alert icmp any any -> any any (msg:"ICMP Photuris Code 0 (Reserved)";
itype: 40; icode: 0;)
alert icmp any any -> any any (msg:"ICMP Photuris Code 1 (Unknown
Security Parameters Index)"; itype: 40; icode: 1;)
alert icmp any any -> any any (msg:"ICMP Photuris Code 2 (Valid
Security Parameters, But Authentication Failed)"; itype: 40; icode: 2;)
alert icmp any any -> any any (msg:"ICMP Photuris Code 3 (Valid
Security Parameters, But Decryption Failed)"; itype: 40; icode: 3;)
```

```
alert icmp any any -> any any (msg:"ICMP Photuris (Undefined Code!)";
itype: 40;)
alert icmp any any -> any any (msg:"ICMP Unknown Type";)
```

For corrections/additions/suggestions for this research paper, please send email to ofir@sys-security.com. Further Information and updates would be posted to http://www.sys-security.com.

Thank you for reading

Ofir Arkin

Founder

The Sys-Security Group

http://www.sys-security.com
ofir@sys-security.com

EXHIBIT 'G'

Welcome to Sys-Security.com

Home

Sys-Security Group

BECAUSE SECURITY IS NOT TRIVIAL

Sys-Security.com is a web site dedicated to computer security research. It is the home of the "ICMP Usage in Scanning" research project.

Navigation
Advisories
Articles Mentioned
Conferences
News
Pictures
Published Papers
Upcoming Events

Books
Links
Tools

Projects
The ICMP Project

VoIP
Xprobe

Latest News

September 19, 2002
"The Trivial Cisco IP Phones Compromise" paper was released...

August 20, 2002
Advisory: More Vulnerabilities with Pingtel xpressa SIP-based IP Phones...

August 9, 2002
Xprobe2 version 0.1 release candidate 1 has been released...

The Trivial Cisco IP Phones Compromise

The following paper lists several severe vulnerabilities with Cisco systems' SIP-based IP Phone 7960 and its supporting environment. These vulnerabilities lead to complete control of a user's credentials, the total subversion of a user's settings for the IP Telephony network, and the ability to subvert the entire IP Telephony environment. Malicious access to a user's credentials could enable "Call Hijacking", "Registration Hijacking", "Call Tracking", and other voice related attacks. The vulnerabilities exist with any deployment scenario, but this paper deals specifically with large scale deployments as recommended by Cisco.

Full Details in PDF format ~200kb
Full Details in PDF Zipped format

More Vulnerabilities with Pingtel xpressa SIP-based IP Phones

Pingtel (http://www.pingtel.com) develops intelligent Java-based voice-over-IP phones and softphones for service providers and enterprises.

Using the vulnerabilities enumerated within this advisory it is possible to jeopardize critical telephony infrastructure based on Pingtel's xpressa SIP-based IP phones and softphones. Additionally, certain vulnerabilities allow an attacker to take complete control over an IP Phone or a softphone node either directly or by circumventing other SIP entities on the network by abusing the 'node's credentials'.

The most severe issue discussed is the way an attacker can exploit vulnerabilities with MyPingtel portal (http://my.pingtel.com) to subvert a VoIP infrastructure which includes IP Phones and/or softphones from Pingtel... >>

Full Details in PDF format ~600kb
Full Details in HTML format
Moderated version in txt format

Xprobe2

Xprobe2 is an active operating system fingerprinting tool with a different approach to operating system fingerprinting. Xprobe2 rely on fuzzy signature matching, probabilistic guesses, multiple matches simultaneously, and a signature database.

The tool's release is accompanied by a white paper explaining our fuzzy approach to operating system fingerprinting and the various problems facing other remote active operating system fingerprinting tools available today, which we have tried to remedy.

The current development code (xprobe2 0.1 rc1) is available from:
http://www.sys-security.com/archive/tools/xprobe2/xprobe2-0.1rc1.tar.gz

The White Paper is available from:
http://www.sys-security.com/archive/papers/Xprobe2.pdf

Xprobe

Navigate Sys-Security.com:
Choose one:
Go

ICMP Usage In Scanning

The ICMP protocol may seem harmless at first glance, but in terms of security ICMP is one of the most controversial protocols with the TCP/IP suite. The risks involved in implementing the ICMP protocol in a network regarding scanning are the subject of this research. Issues discussed range from understanding the different ICMP messages, reconnaissance (host detection, inverse mapping, Active and passive operating system fingerprinting) and more >>

For corrections/additions/suggestions for this page, please send email to:

Trace-Back

Trace-Back is "*A Concept for Trac[ ]nd Profiling Malicious Computer Attackers*"
From the Introduction:

"In the computer security arena, every now and then, a vulnerability comes along causing a significant impact. The impact of a vulnerability is based on factors such as popularity of the vulnerable platform and the ease of exploitation of the vulnerability. Lots of research gets done on a vulnerability, beginning from its origin to the various permutations and combinations of exploit code that come out subsequently. In recent years, we have seen self-propagating exploit code (in other words, worms) becoming quite popular.

Very little is known about the events taking place in the time period between the instance that a vulnerability gets discovered by an individual or a small group of individuals, and the moment when exploit code becomes publicly available on the Internet. To zero in on the origins of a particular piece of exploit code is quite a daunting task. Very little research has been done on the subject outside of government or military organizations. Tracing back origins is a very tricky task, especially if one has to reconstruct events backwards. This paper addresses this very issue - trying to roll the film reel backwards from the time the exploit code becomes widespread in public, and filling in the blank frames to the beginning of the movie. This may not be the ultimate "big-bang" theory of the exploit universe, but it provides us with new viewpoints on exploits and their originators..."